THE TECHNICAL COOPERATION PROGRAM

SUBCOMMITTEE ON NON-ATOMIC MILITARY RESEARCH AND DEVELOPMENT

PROCEEDINGS OF THE TTCP WORKSHOP ON SOFTWARE METRICS

ROCHESTER INSTITUTE OF TECHNOLOGY
ONE LOMB MEMORIAL DRIVE
ROCHESTER, NY

MAY 21 - 24 1990

Hosted by

U.S. Air Force Systems Command
Rome Laboratory

DISCLAIMER

The citation of trade names and names of manufactures in this report is not to be construed as official Government endorsement or approval of commercial products or services references herein.

This page intentionally left blank.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Technical Report - 5/21/90 to 5/24/90

4. TITLE AND SUBTITLE

Proceedings of the TTCP Workshop on Software Metrics

5. FUNDING NUMBERS

6. AUTHOR(S)

William Agresti, Joseph P. Cavano, Robert Converse, Janet Dunham

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rome Laboratory
RL/C3C
Griffiss AFB, NY 13441

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

same

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

same

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document presents the findings of the TTCP Workshop on Software Metrics held on 21-24 May 90 at Rochester Institute of Technology in Rochester, NY. The workshop was hosted by the Air Force Systems Command's Rome Laboratory and sponsored by The Technical Cooperation Program (TTCP). The workshop brought together experts from gov't, industry and academia to examine the state-of-the-art and state-of-the-practice of software measurement technology and to share ideas and develop recommendations for further development and application of software metrics. Four panels were convened to look at four different but overlapping aspects of software metrics, namely software product metrics, software process metrics, system metrics and software metrics in acquisition. The findings of the panels is presented in this proceedings.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software measurement, software metrics, product metrics, process metrics, software engineering	156
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCL	UNCL	UNCL	U/L

This page intentionally left blank.

OFFICE OF THE DIRECTOR OF
DEFENSE RESEARCH AND ENGINEERING

WASHINGTON, DC 20301

FOREWORD

The Technical Cooperation Program (TTCP), Technical Panel on Software Engineering (XTP-2) extends its gratitude to the U.S. Air Force Rome Laboratory (formerly Rome Air Development Center) for its support of TTCP initiatives in software engineering research and development. By providing the resources to make this Software Metrics Workshop possible Rome Laboratory not only has contributed to the advancement of knowledge in this technology area, but also has further fostered cooperation and collaboration among the TTCP international community. The support and dedication of Rome Laboratory staff in planning, organizing and hosting this workshop was a significant factor in ensuring the Workshop's success from the outset. XTP-2 is also particularly grateful to the Workshop General Chairperson, the Workshop Panel Chairpersons, and all the government, industry and academic Workshop participants who worked long and late to contribute their expertise to make the outcome successful in every way. We hope that the effort was as enjoyable and beneficial for all the participants as it was for the TTCP XTP-2 Panel Members.

The Workshop objectives, to survey and evaluate the state-of-the-art, the state-of-the-practice, technical issues and technology gaps in software metrics, and, to recommend courses of action, have been fulfilled, and are recorded in these Proceedings. These findings and recommendations are now under review by TTCP XTP-2 panel members and other representatives of the member TTCP Governments for consideration of appropriate follow-on actions within their individual countries.

The TTCP XTP-2 Panel members recognize the potential for achieving important improvements in software engineering and software program management through the use of software metrics. They will continue to assess the state-of-the-art and -practice, make appropriate recommendations promoting their application, and support further research and development.

Joseph C. Batz
Chairman, TTCP XTP-2

This page intentionally left blank.

TTCP WORKSHOP ON SOFTWARE METRICS

MAY 21-24 1990

Rochester Institute of Technology
One Lomb Memorial Drive
Rochester NY 14623-0887

WORKSHOP LEADERS

Mr Samuel DiNitto	Workshop Chairperson
Ms Janet Dunham	Panel I Chairperson "Software Product Metrics"
Dr William Agresti	Panel II Chairperson "Software Process Metrics"
Mr Robert Converse	Panel III Chairperson "Software Metrics Acquistion Issues"
Mr Joseph P Cavano	Panel IV Chairperson "System Metrics"

THE TECHNICAL COOPERATION PROGRAM
Technical Panel Number 2 on Software Engineering (XTP-2)

Mr. Joseph Batz, United States National Leader & Chairperson

Mr. Jean-Claude Labbe, Canadian National Leader

Mr. Michael J. Looney, United Kingdom National Leader

Mr. Stefan Landherr, Australian National Leader

Dr. Martin I. Wolfe, United States Army, CECOM
Mr. Phillip Andrews, United States Navy, SNWSC
Mr. Thomas P. Conrad, United States Navy, NUSC
Mr. Samuel DiNitto, United States Air Force, RL
Mr. Robert Harris, United States Air Force, WL

This page intentionally left blank.

TABLE OF CONTENTS

1. **EXECUTIVE SUMMARY** 1
1.1 Introduction 2
1.2 Software Product Metrics 2
1.3 Software Process Metrics 3
1.4 Software Metrics Acquisition Issues 3
1.5 System Metrics 4

2. **WORKSHOP CHARGE** 7

3. **WORKSHOP PROCEEDINGS** 11
3.1 Introduction 13

3.2 **Panel 1**
Software Products Metrics 15

3.2.1 GENERAL INFORMATION 15
3.2.1.1 Panel Participants 15
3.2.2 INTRODUCTION AND OVERVIEW 16
3.2.2.1 Properties of Product Metrics 16
3.2.2.2 Validation of Product Metrics 17
3.2.2.3 Framework for Viewing Product Metrics 17
3.2.2.3.1 Software/System Properties Framework 18
3.2.2.3.2 User Framework 18
3.2.2.3.3 Life Cycle Framework 19
3.2.2.3.4 Quality Gold Framework 20
3.2.2.4 Relationship of Product Metrics to Process & System Metrics 20
3.2.3 STATE-OF-THE-PRACTICE 21
3.2.3.1 Cost 22
3.2.3.2 Benefits 22
3.2.3.3 Problems 23
3.2.3.4 General Issues 23
3.2.4 STATE-OF-THE-ART 24
3.2.4.1 Approaches 24
3.2.4.2 Use Scenarios 24
3.2.4.3 Experiences to Date 25
3.2.4.4 General Issues 26
3.2.4.5 Know Future Directions 27
3.2.5 RECOMMENDATIONS 27
3.2.6 REFERENCES 31

3.3 **Panel 2**
Software Process Metrics 32

3.3.1 GENERAL INFORMATION 32
3.3.1.1 Panel Participants 32
3.3.2 INTRODUCTION 33
3.3.3 STATE-OF-THE-PRACTICE 34
3.3.3.1 What process metrics are being used in local or global settings? 34
3.3.3.2 What is industry's experience with the cost of metrics programs? 35
3.3.3.3 What problems are encountered in process metrics programs? 36

3.3.3.3.1 No sound theoretical measurement methodologies exist 36
3.3.3.3.2 Data Collection 36
3.3.3.3.3 Data Analysis, Interpretation and Validation 36
3.3.3.3.4 Tools 36
3.3.3.3.5 Management 36
3.3.3.3.6 User Resistance 37
3.3.3.3.7 Institutional Resistance 37
3.3.3.4 What are the benefits of process metrics programs? 37
3.3.3.5 Why are process metrics being used? 38
3.3.3.6 What examples of state-of-the-practice use? 39
3.3.3.7 What are general issues associated with state-of-the-practice process metrics? 40
3.3.3.7.1 Big-P versus Little-p 40
3.3.3.7.2 Applying measurement technology to software 40
3.3.3.7.2.1 The scales used for metrics 40
3.3.3.7.2.2 The terminology used in reference to metrics 40
3.3.3.7.2.3 The proliferation of metrics and the inherent problem of overlapping metrics 40
3.3.3.7.2.4 The abstraction level of metrics 41
3.3.3.7.2.5 Reuse of metrics 41
3.3.3.7.2.6 Impact of the development process model 41
3.3.3.7.2.7 Validation of metrics 41
3.3.3.7.2.8 Contractual Issues 41
3.3.4 STATE-OF-THE-ART 41
3.3.4.1 What approaches characterize state-of-the-art process metrics programs? .. 41
3.3.4.2 What are state-of-the-art use scenarios for process metrics? 42
3.3.4.3 What are experiences to date with state-of-the-art process metrics? 43
3.3.4.4 What are general issues associated with state-of-the-art process metrics? ... 44
3.3.5 RECOMMENDATIONS 45
3.3.6 REFERENCES 47

3.4 **Panel 3**
Software Metrics Acquisition Issues 49

3.4.1 GENERAL INFORMATION 49
3.4.1.1 Panel Participants 49
3.4.2 INTRODUCTION AND OVERVIEW 50
3.4.2.1 Scope 50
3.4.2.2 Purpose 50
3.4.3 STATE-OF-THE-PRACTICE 50
3.4.3.1 Contract Strategies 50
3.4.3.1.1 Contract Types 50
3.4.3.1.2 Incentives 51
3.4.3.1.3 Current Methods for Paying Incentives 51
3.4.3.1.4 Selection Strategy 51
3.4.3.1.5 Contractor/Sub-contractor Relationship 51
3.4.3.2 Standards, Policies and Guidelines 52
3.4.3.3 Experiences 54
3.4.3.3.1 Cost 54
3.4.3.3.2 Benefits 54
3.4.3.3.3 Problems 55
3.4.3.3.4 Lessons Learned 56
3.4.4 STATE-OF-THE-ART 56
3.4.4.1 Emerging Contract Strategies 56

3.4.4.1.1 Contract Types 57
3.4.4.1.2 Incentives 57
3.4.4.1.3 Metrics for "paying" Incentives 58
3.4.4.1.4 Selection Strategy 58
3.4.4.1.5 Contractor/Sub-contractor Relationship 58
3.4.4.2 Standards, Policies and Guidelines 58
3.4.4.2.1 Issues Regarding Applications of Standards/Guidelines 59
3.4.4.3 Know Future Directions 59
3.4.5 GENERAL ISSUES 59
3.4.5.1 Requirement Change 60
3.4.5.2 Validation of Metrics 60
3.4.5.3 Cost of Metric Data Collection 60
3.4.5.4 Time Lag for Reporting Metric Data 60
3.4.5.5 Protection of Sensitive Data 60
3.4.5.6 Personnel Career Development 60
3.4.5.7 Uses of Acquisition Metrics 60
3.4.6 RECOMMENDATIONS 61
3.4.6.1 General 61
3.4.6.2 Acquisition Approaches 61
3.4.6.3 Validation Approaches for Future and State-of-the-Art Metrics 63
3.4.6.4 Technology Programs 63
3.4.6.5 Other 64

3.5 **Panel 4**
System Metrics 65

3.5.1 GENERAL INFORMATION 65
3.5.1.1 Panel Participants 65
3.5.2 INTRODUCTION AND OVERVIEW 66
3.5.3 STATE-OF-THE-PRACTICE 67
3.5.3.1 Maintainability 68
3.5.3.2 Performance 68
3.5.3.3 Extendability 68
3.5.3.4 Safety 69
3.5.3.5 Useability 69
3.5.3.6 Cost 69
3.5.3.7 Schedule 69
3.5.4 METRICS FOR SOFTWARE QUALITY 70
3.5.4.1 Reliability 71
3.5.4.2 Maintainability 72
3.5.4.3 Digital Performance 73
3.5.4.4 Extendability 73
3.5.4.5 Safety 74
3.5.4.6 Useability 74
3.5.5 CONCEPTUAL MODEL FOR SYSTEMS METRICS 74
3.5.5.1 System Definition/System Architecture in Preliminary Design 74
3.5.5.2 Conceptual Prototype Used Throughout Acquisition 76
3.5.5.3 Engineering and Product Planning/Hardware and Software Functional Allocation 77
3.5.5.4 System Integration and Testing 78
3.5.5.5 Test and Evaluation 78
3.5.5.6 User Acceptance 79
3.5.5.7 Deployment 79
3.5.6 SUMMARY: RECOMMENDATIONS FOR SYSTEM METRICS 79

3.5.7 REFERENCES 81
3.5.8 APPENDIX I: MTBF as a System Metric: A Negative View 82
3.5.9 APPENDIX II: System Definition Languages in System Software Definition 83

4. **BIBLIOGRAPHY** 85

LIST OF APPENDICES

APPENDIX A .... Workshop Agenda
APPENDIX B .... List of Attendees
APPENDIX C .... Presentation Slides

LIST OF FIGURES

Figure 1 Conceptual Model for System Metrics 75

LIST OF TABLES

Table 1 Acquisition View of Reliability 19
Table 2 Products Currently Analyzed by Software Product Metrics 25
Table 3 Factor Orientation to System, Hardware and Software 70

1

EXECUTIVE SUMMARY

1. EXECUTIVE SUMMARY

1.1 INTRODUCTION

A key element to understanding, managing and controlling the development of today's massive, complex and sophisticated system is information. Information about the system - its purpose, functionality, quality characteristics and how well it does what it was supposed to do. Also, information about the process and people producing the system - the procedures, tools and methods used, progress, schedule and resources expended, cost and personnel capabilities. Information that, in essence, are measurements of the system and the process. As Sintic and Joiner stated in their paper Managing Software Quality [1], "A fundamental management principle is that a quality or a process can be controlled only if it can be measured." Measurements are essential to provide the visibility needed to properly manage, evaluate and control large, complex programs.

The TTCP Workshop on Software Metrics was convened to explore the state and future of measurement technology, particularly the concepts, problems and issues that pertain to software systems and software system development. The workshop was hosted by the US Air Force System Command's Rome Laboratory at the facilities of Rochester Institute of Technology in Rochester, New York on May 21-24, 1990. The Technology Cooperation Program (TTCP) Panel on Software Engineering (XTP-2) sponsored the workshop.

Thirty-nine (39) international scientists and engineers participated in the Software Metrics Workshop. The participants, many of whom are well recognized for their experience and expertise in software system development and software engineering, brought with them a wealth of knowledge gained from years of involvement in the field in government, industry and academia.

This proceedings documents the findings and recommendations of the workshop and its panels.

1.2 SOFTWARE PRODUCT METRICS

The panel developed a common understanding of product metrics, discussed the state-of-the-practice in using product metrics including their benefits and problems, and addressed issues critical to improving the state-of-the-art. These discussions resulted in the recommendation of ten (10) courses of action.

The panel recommended research and development be conducted for the following: early life-cycle metrics, product assessment metrics, metrics for characterizing software usage, and metrics for trustworthy and other high integrity applications. The panel also recommended the development of a standard scheme for classifying metrics, a mechanism for effective reporting of software product quality metrics, and a technology transfer mechanism to promote use and acceptance of metrics.

Finally, recommendations were proposed to investigate cost/benefits of using metrics; to refine, develop, and investigate tools to support product measurement of languages and architectures; and to conduct research to integrate product, process, system and acquisition metrics.

1.3 SOFTWARE PROCESS METRIC

The Software Process Metrics panel began its session by agreeing to use the definition of process proposed by Watts Humphrey in his book Managing the Software Process [2], "Process is a set of activities, methods, and practices that are used in the production and evolution of software."

The panel addressed the state-of-the-practice in the use of process metrics, the state-of-the-art including process metrics projects in various stages of research and development, and identified five (5) recommendations to advance the state of process metrics.

The panel recommended two research programs be undertaken. One, to develop metrics maturity framework and another to validate promising state-of-the-art process metrics technology. Other recommendations include: promoting process improvements and its integration with a metrics program; promoting software work breakdown structures that reflect the process as defined by the developer; and the establishment of me'rics education and training courses.

1.4 SOFTWARE METRICS ACQUISITION ISSUES

The panel addressed issues concerning the use of metrics up to the time of contract award and the contractual mechanisms needed to ensure that metrics data is obtained and used during the period of contract performance. The panel defined three areas to address: the use of metrics to develop better criteria for evaluation of proposals and assessment of the offeror's abilities; contractual mechanisms required to obtain and analyze metrics for effective management and monitoring of the process and evaluation and assessment of the products; and recommendations of acquisition strategies and metrics applicable to the acquisition process.

The panel cited the need for a fundamental change to the acquisition process. This change, the panel stated, is required to decouple the build-the-system phase of the acquisition from the original cost estimate to allow for changes in requirements and design phases.

On acquisition approaches, recommendations were made to: base milestone completion on quality criteria rather than schedule or cost; use objective incentive type contracts rather than fixed- fee (FF) or subjective award fee; use the Process Maturity Levels from SEI Contractor Assessment Methodology for source selection and to track contractor maturity growth; and split the acquisition into two distinct phases - a Cost-Plus-Incentive-Fee (CPIF) type contract for the requirements and design phase, and CPIF or Fixed-Price-Incentive-Fee (FPFF) type contract for the build phase.

Regarding validation of future and state-of-the-art metrics, the panel recommended: establishment of metric data collection standards, contract language and a central repository for storage, analysis and distribution of summary information; and that case studies and controlled experiments be conducted on large acquisitions.

The panel recommended research and technology programs to: develop an integrated set of acquisition package items to insure that the structure and conduct of the acquisition can take advantage of risk reduction approaches including use of metrics; develop guidelines/handbooks for collection and analysis of metrics; develop clear guidelines for tailoring metrics; develop criteria and metrics that can be used to measure the quality of request for proposals (RFP) and acquisition packages; and develop metrics and analysis methods that can be used during the requirements and design phases of an acquisition to measure both the quality of the requirements and design and their rates of change.

Other recommendations included: provide training to program, contract and management personnel in proper use and interpretation of process and product metrics and proper application of the "modified acquisition approach"; ensure mandated software metrics collection and dissemination rules are consistent with rules governing cost/schedule reporting; develop and use criteria and metrics to clarify use of commercial-off- the-shelf (COTS) products as a part of software acquisition; and develop metrics to measure changes in requirements that affect acquisition.

1.5 SYSTEM METRICS

The System Metrics panel was tasked to look at metrics at the system level. The panel took the big picture view of a project. Their objective was to describe metrics which characterize a system as opposed to metrics which characterize a system's constituent parts. The panel also focused on metrics and issues that were generic in that they could relate to any system and not be limited to a project or specific application. The panel identified a total of twenty-one (21) recommendations.

Regarding current practices, the panel recommended that system quality requirements for reliability, maintainability, digital performance (which is defined as the product of the processing rate and algorithm efficiency), safety and usability be identified in the request for proposal (RFP) and should reflect the customers inputs. Concerning their Conceptual Model of System Metrics, a similar recommendation was made that projects should identify the system's quality requirements necessary to complete a given mission.

For software quality, the panel recommended research programs to determine a complete set of system quality factors; establish relationships between software metrics and percent reliability and combine these metrics with hardware reliability figures of merit; develop global metrics for hardware and software reliability; develop a software reliability metric compatible with Mean-Time-Between-Failure (MTBF); and develop acquisition guidelines that define high level software metrics that are suitable for combining with hardware metrics. The panel also recommended that software reliability budgets be allocated to software sub-systems during system design as is done during hardware design; and that when tracking maintenance activities, distinctions should be made between software error fixes and system improvements/enhancements.

Regarding hardware/software performance, the panel recommended the use of a model/benchmarks to capture the interrelationships of the hardware and software to performance and the other "ilities" be investigated.

For system definition and system architecture, recommendations were to: investigate Total Quality Management (TQM) techniques, such as Quality Function Deployment (QFP) for use in translating customer requirements into software specifications, design and reliability requirements, and Statistical Process Control (SPC) for monitoring and controlling specific products/process metrics in the software development environment. The panel also recommended: study of software module definition methods which can reduce failures caused by module-to-module incompatibilities; and research to determine appropriate system definition tasks for real-time, distributed and parallel processing systems.

In their discussion of the benefits of prototyping, the panel recommended that prototypes be used throughout the life-cycle to demonstrate quality requirements as well as functional requirements and to explain system changes and enhancements after deployment.

Other recommendations are as follows: investigate use of reliability warranties/guarantees for software; collect sufficient data during operational use to provide meaningful determination of the level achieved for specified system factors and to prepare cause and effect diagrams; study utility of system definition languages for unambiguous and complete definition of functions implemented in software; and develop and use very high level languages to provide ways to define system functions.

1.6 REFERENCES:

[1] J. H. Sintic; H. F. Joiner; "Managing Software Quality," In Journal of Electronic Defense, May 1989.

[2] W. Humphrey; "Managing The Software Process," Reading, Massachusetts: Addison-Wesley; 1989.

This page intentionally left blank.

2

WORKSHOP CHARGE

This page intentionally left blank.

2. WORKSHOP CHARGE

By: Samuel A. DiNitto, Workshop Chairperson

The workshop participants were specifically charged with several tasks with regard to several different but overlapping aspects of software metrics. The aspects were concerned with product metrics, process metrics, system metrics, and software metrics in acquisition, and each of these aspects was to be the concern of a single panel.

The first task for the Product, Process, and System Metrics Panels dealt with capturing the state-of-the-practice. This entailed identifying what was being used routinely in local or global settings; relating experiences with regard to cost, benefit, and problems; determining why the metrics were being used; and relating any general issues. For this task, the Software Metrics in Acquisition Panel was to focus on the state-of-the-practice contract strategies in the form of incentives, contract types, etc.; contractor/sub-contractor relationships; standards, policies, and guidelines; experiences (cost, benefits, problems, lessons learned, etc.); the rationale; and any general issues they could identify.

The second task for the Product, Process, and System Metrics Panels asked them to address the state-of-the-art with respect to approaches, use scenarios, experiences to date, general issues, and known future directions. The Software Metrics in Acquisition Panel was requested to address the same areas and in addition to identify any promising or emerging approaches to software metrics not in routine use.

Finally, all panels were instructed to produce recommended courses of action. These recommendations were to take the form of possible (joint) technology programs, validation approaches for future as well as state-of-the-art metrics, and acquisition approaches, to include strategies for expanded use, "incentivizing" maturity models, etc.

This page intentionally left blank.

3

WORKSHOP PROCEEDINGS

This page intentionally left blank.

3. WORKSHOP PROCEEDINGS

3.1 INTRODUCTION

Containment of escalating costs of computer software in the DoD has become a major concern as more and more of the DoD budget is consumed acquiring and maintaining its enormous inventory of sophisticated software systems. These costs, according to recent estimates, will exceed $26B in FY91. Moreover, the cost of maintaining its software is increasing at an alarming rate of 26% per year.

The growing concern over spiralling software costs has spawned numerous programs and initiatives within the DoD and the software industry aimed at improving the way software systems are acquired, developed and maintained. Numerous diverse solutions to this multi-facetted problem are being pursued with the mutual objectives of increasing the quality of software systems and improving, by orders of magnitude, the productivity and life-cycle costs of software acquisitions.

Technologists, researchers and developers are attacking the problem from many directions but their programs share a common activity which this workshop has chosen as its theme - software measurement. In software acquisition and software technology development, measurement takes many forms and serves many purposes. It is a means for determining progress, results and evaluating new and emerging software engineering technology. It is a means for quantitatively assessing and evaluating software systems and software products. And, it is a mechanism for providing program visibility essential for proper management and control of large system procurements.

As meaningful and pervasive as measurement activities are in the software field, software measurement as a technology is fragmented and immature. The workshop was held to examine some of the problems, differences and possible benefits of software measurements and to obtain expert guidance and advice regarding the use, improvement, and future direction of the technology.

The TTCP Software Measurement Workshop brought together experts from government, industry and academia, with expertise in software development, program management, software engineering and software measurement technology. Their task was to examine the state-of-the-art and the state-of-the-practice of software measurement technology, to discuss and share their knowledge and related experiences, and to provide recommendations to improve and advance the development and utilization of the technology.

The workshop was hosted by the US Air Force System Command's Rome Laboratory at the facilities of Rochester Institute of Technology in Rochester, New York on May 21-24, 1990. The Technology Cooperation Program (TTCP) Panel on Software Engineering (XTP-2) sponsored the workshop.

The Rome Laboratory is the Air Force's center of expertise for research and development in command, control. communications, and intelligence (C3I) and related technologies. The Laboratory's C2 Software Engineering Division is responsible for the development of advanced software engineering technology needed to ensure the success of C3I systems. The Division's software engineering program encompasses the total life-cycle with emphasis on software support environments, system definition technology, software and system quality and software for high performance architectures.

The TTCP is a formal arrangement for mutual sharing of research and development resources/tasks established by member country foreign and defense ministries. Member countries include Australia, Canada, New Zealand, the United Kingdom, and the United States. Within the structure of the TTCP, there are eleven (11) subgroups made up of forty-four (44) working panels and twenty-two (22) action groups. The TTCP/XTP-2 Panel is concerned with the creation and life cycle support of software for defense related applications.

Thirty-nine (39) international scientists and engineers participated in the Software Metrics Workshop. The participants, many of whom are well recognized for their experience and expertise in software system development and software engineering, brought with them a wealth of knowledge gained from years of involvement in the field in government, industry and academia.

The participants were divided into four (4) groups or panels. The panels were tasked to look at the state-of-the-art, the state-of-the-practice and future directions of metrics in pre-selected task areas., Panel 1 addressed Software Products Metrics and was chaired by Ms Janet Dunham of Research Triangle Institute. The second panel, chaired by Dr. William Agresti of the MITRE Corp., was assigned Software Process Metrics. Panel 3 addressed Software Metrics Acquisition Issues and was chaired by Mr. Robert Converse of the Computer Sciences Corp. And, the fourth panel, chaired by Mr. Joseph Cavano of Rome Laboratory, focused on System Metrics.

The workshop was chaired by Mr. Samuel DiNitto of Rome Laboratories. Mr. DiNitto is the chief of the C2 Software Technology Division.

This proceedings documents the findings of the workshop and its panels.

3.2 Panel 1: Software Product Metrics

3.2.1 General Information

3.2.1.1 Panel Participants

NAME	EMPLOYER	COUNTRY
Janet Dunham, Chair	Research Triangle Institute	USA
David Budgen	University of Stirling	Scotland
Andrew Chruscicki	US Air Force/Rome Laboratory	USA
Roger Dziegiel, Jr	US Air Force/Rome Laboratory	USA
Walter Ellis	IBM	USA
Stefan F. Landherr	DSTO/Electronics Research Laboratory	Australia
John McDermott	US Navy/Naval Research Lab	USA
Glenn Racine	US Army/AIRMICS	USA
Carl Seddio	Eastman Kodak Corp.	USA
Christopher Smith	Software Engineering Res. Center	USA

3.2.2 Introduction and Overview

The TTCP Product Metrics panel developed a common understanding of product metrics, discussed the state-of-the-practice in using product metrics, and addressed issues critical to improving the state-of-the-art. These discussions resulted in the panel recommending ten courses of action.

Product metrics provide a set of abstract measures of the software product, which is itself an abstraction. Different product metrics are used to assess a range of "properties" of a software product. Examples of product metrics include McCabe's source code complexity measure [1], Henry and Kafura's information flow metric 12], and the Rome Laboratory's reliability prediction figure-of-merit [3]. The panel discussed desirable properties of product metrics, metric validation issues, current frameworks for viewing product metrics, and the relationship between software product metrics and other metrics.

3.2.2.1 Properties of Product Metrics

To be useful, a product metric needs to possess some important properties. These include:

- being stable or robust (i.e., the metric cannot be easily affected by adding null statements to a procedure);

- representing the properties of a system at a particular time (i.e. forming a "snapshot" of the state of the system at that time);

- providing a measure that reflects a particular viewpoint of the system (i.e. reflects particular properties and relationships).

To achieve these properties, some important issues for software product metrics must be addressed. These issues include what basic measures should be used and what software product metrics should be derived from them, how to unambiguously extract these measures from abstract representations, and how to deal with a lack of any absolute interpretation.

A fundamental property of making measurements is countability. That is, measurement consists of counting "entities", whether these be units on a ruler, clock pulses or lexical tokens in a program. The basic count may itself provide a useful measure of this, as for example, when we use a ruler to measure length. Often however, we use the basic measures and combine these to derive further measures that reflect the properties of other abstractions that may be useful. An example of combining measures is combining measurements of time and position in order to calculate energy. In a similar fashion, we combine various software product measures in order to provide metrics of such software properties as complexity, reliability, etc.

The various measures can be considered as being either primitive or derived metrics. Because software is an abstraction, we can identify only a limited set of directly countable properties - which is further complicated by the need to extract these from a representation such as code, diagrams etc. Our ability to extract these properties in an unambiguous form limits what is possible for software product metrics. An example of this problem is the difficulty of defining exactly what is meant by Source Lines of Code (SLOC) for a particular source language.

A further issue for software product metrics is the lack of any absolute interpretation for the metric values themselves. For most purposes, we are more concerned with comparing values for a metric (e.g., this solution is more/less complex than that one) and need to consider relative values of a metric. A further useful concept is that of a tolerance, by which we can specify the range of acceptable values for a particular metric. To some extent, such uses for metric values reflect the nature of the design process itself, which involves making choices with a potentially large "solution space".

3.2.2.2 Validation of Product Metrics

Because software product metrics are not absolute in any sense, any form of metric validation process is likely to be domain-specific. In comparison with physics/engineering, there is currently no platinum-iridium bar or "standard" wavelength to compare against in order to provide a means of calibrating or "normalizing" our measurements. So for purposes of any validation, we need to establish:

- the problem space/domain of concern,
- the interpretation of the metric in terms of that domain,
- whether there are any exceptional conditions that need to be noted (i.e., situations where the structure of the code may give rise to misleading values), and
- the criteria to be used to determine whether or not the validation proceeds is successful.

The domain-specific nature of the process also needs to take into account a set of possibly significant factors, which can include the programming language, the type of problem, and the design process used. These factors also influence the way that the values of a metric are interpreted by the end user [4] . A consequence of these points is that it is possible for a metric to be partially validated, indicating that its usefulness/relevance has been demonstrated for a particular type of application only. An example of this might be McCabe's Cyclomatic Complexity Metric- this was prototyped using FORTRAN programs, the recommended bounds of acceptability might be quite different if the language used was for example, Ada, and might be further influenced by the problem, too. For that particular example it might also be significant to consider whether or not CASE statements were in use, since the treatment of these difficult to manage in a consistent form. For many metrics, the relationships being measured may be relatively language independent, however, it is difficult to identify what meaning we could apply to McCabe's metric if used with a declarative programming language.

3.2.2.3 Frameworks for Viewing Product Metrics

Currently, there are several ways of viewing software product metrics. These views consider:

- the properties of the system being measured,
- the users of the metrics and their organizational role,
- when the metrics are being used throughout the life cycle, and

• software quality goals.

Frameworks addressing each of these views are briefly discussed in the following paragraphs.

3.2.2.3.1 *Software/System Properties Framework*

The design of software often makes use of structural, behavioral, and functional viewpoints for describing the properties of a system. Each of these viewpoints reflects particular relationships between the entities of the system, so it is useful if we can identify a set of metrics that reflects these viewpoints, since these will be used by the developer at various stages beyond that of design.

Structural properties are those which quantify those relationships which reflect the static architectural features of a system. Such relationships include those which define scope, packaging, invocation hierarchy, etc. Examples of metrics that can be used to measure such properties include Halstead's measures such as "Volume", vocabulary, etc. [5], McCabe's Cyclomatic Complexity measure [1], Henry & Kafura's Information Flow Metric [2]. These metrics can all be applied to code, since this is essentially a static description.

Behavioral properties are concerned with causality relationships and so reflect a dynamic view of a system. From a design point of view, these relationships can be captured by such forms as "State Transition Diagrams" or "State charts". While in principle these can be (at least partly) derived from analysis of code, no obvious examples can be identified.

Functional properties are concerned with the operations that are performed by a system. These are again dynamic, and are not easily captured in any abstract form.

The metrics in current use are essentially concerned with structural properties, since these are static and are relatively easily derived from analysis of code. For both the behavioral and functional viewpoints, the lack of a firm representation, and the difficulty of quantifying the temporal aspects make metrics difficult to apply.

3.2.2.3.2 *User Framework*

The purpose of using software metrics is to provide insight and feedback. The nature of the software metrics used by a person, and the manner in which they are used, depends on the role of that person in the software acquisition/development organization. Users of software product metrics include acquisition managers, development managers, technical developers, certifiers, owners, and end-users.

The need for providing information about objectives in a user framework is illustrated by taking the behavioral characteristic of reliability and addressing views from the technical, acquisition, and management perspectives.

A technical view of reliability could contain the following information:

- METRIC Life Cycle Object

- Requirements traceability

- Requirements, design, code, test
- Uses Hierarchy design, code
- Cyclomatic Complexity Design, Code
- Defect Density Early Inspection Faults
- Mean-Time-To-Discover-Next K Faults
- Later Test Faults
- Mean-Time-To-Failure
- Failure Rate Failures

An acquisition view of reliability could contain the information found in Table 1 drawn from [6] .

Application Level	Terminology	Metric Worksheet	L-C Activity	L-C Phase
System	System Function CSCI	0	System, Software Requirement Analysis	Demonstration and Validation
CSCI	CSCI Software Function	1	Software Requirement Analysis	Full Scale Development
CSCI	CSCI Top-Level CSC	2	Preliminary Design	Full Scale Development
CSCI	CSCI Top-Level CSC; Lower Level CSC; Unit	3A; 4A	Detailed Design Code and Unit Test	Full Scale Development
Unit	Unit	3B; 4B	Detailed Design Code and Unit Test	Full Scale Development

Table 1. Acquisition View of Reliability

A manager's view of reliability could include all of the above plus resources including cost, labor, computer time, computer storage, training.

3.2.2.3.3 *Life Cycle Framework*

Most software product metrics are tied in some way or other to the software life-cycle. Similarly, system metrics are tied to the system life cycle and software process metrics are closely tied to the software life cycle activities. Thus, the application and interpretation of a process metric depends on the particular life cycle model being followed (waterfall, spiral, etc.) and on the current phase of that life cycle (prototyping, main development, maintenance, etc.)

Product metrics can only be applied when the appropriate products are available (in partially or fully completed form). In all life cycle models, there are logical precedence relationships between software products, such that a dependent product cannot be started until its ancestor product is started, and it cannot be completed until the ancestor product is completed (i.e. start-start and end-end precedence relationships). However, the order in which software products are completed may depend on the life-cycle model being followed, and the exact interpretation of the metric usually depends on whether the product is complete or not.

3.2.2.3.4 *Quality Goal Framework*

Proper selection, application and interpretation of software metrics requires a clear understanding of the purpose and end-use of the measurements and values. The purpose of metrics is to provide insight and feedback, such as whether certain technical criteria are being satisfied, and whether overall goals are being met.

One useful model is the hierarchical framework developed by Rome Laboratory for Management Indicators [7]. This framework specified a number of Software Quality Factors, the Criteria which characterize that factor, and a selection of metrics to measure/test against those criteria. Although the RL Software Quality Framework was devised for quality factors only, it could be extended to cover other factors and other broad types of metrics.

3.2.2.4 Relationship of Product Metrics to Process and System Metrics

Product metrics are concerned with the properties of the software entities rather than with the process from which they were derived. Since the structure of software (as measured by structural metrics) is at least partly determined by the form of the design process, the two do interact in that any assessment of a product metric may need to consider the process as a part of the domain. For example, a design produced by using object-oriented techniques is likely to differ structurally from a design produced using structured design techniques. In applying structural metrics, any interpretation of the results may need to be weighted in order to allow for the influence of the design method.

In a similar way, we can expect that process metrics are also influenced by the nature of the product. This is because both the type of problem, and the constraints upon the solution, may influence the process by effectively limiting the solution space that is available to the designer.

There are a multitude of metrics that could potentially be used for a project or in an organization, but not all of them would necessarily be useful. The life-cycle model being followed has already been mentioned as one determinant of whether a certain metric or class of metrics would be useful in a particular situation. Another important determinant is the maturity of the software process being used in the project or organization. The SEI Process model [8] provides a framework for discussing this issue.

Certain basic product and process metrics are useful at all levels of process maturity. However, some sophisticated metrics would not be appropriate at low levels of process maturity (e.g. SEI levels 1 and 2); the values would not be meaningful in that process context or because the process is not mature enough for the metrics to generate appropriate feedback.

System metrics are concerned with the interactions that occur between the software and the hardware. Issues such as performance, reliability, etc. are of this class since they depend upon both the software structures and also the way that these are mapped on to the hardware.

Software metrics are concerned with the properties of the software when considered in isolation from its eventual environment and any interactions with the environment. For structural metrics this is relatively evident, since they are concerned with static structures only. Since behavioral metrics are essentially dynamic in form, it would appear that these would be more dependent upon the underlying system and the nature of the hardware. However, the behavioral viewpoint is essentially concerned with providing a state-oriented description, rather than with describing such issues as timing. The use of time in this context is only concerned with ordering issues. Similarly, the functional viewpoint is not concerned with interaction between software and hardware, although individual functions may be mapped on to these in different ways.

3.2.3 State-of-the-Practice

The various frameworks for viewing software product metrics encompass much more than what is currently being practiced. In general this practice is focused on the use of software product metrics during coding and testing by software engineers. The trend for the short-term is toward functional metrics on design. And the long-term outlook is for behavioral metrics on requirements. The state-of-the-practice can be summarized as moving from later to earlier stages of development and moving from the concrete to the more abstract product.

The objective of metrics is to provide visibility to developers, maintainers, managers, owners, and users of the developing product and indirectly of the process. Developers and maintainers, given desired behavioral characteristics from requirements, can establish a process strategy to achieve these objectives. This strategy can consist of activities, methods, tools, architectures, and measurement. The elements of this process strategy can be quantitatively monitored during development to measure progress toward quality goals. Management and acquirers can use metrics to provide status for project control, assessment of progress toward completion, and evidence for trading-off risks, benefits and costs. Example metrics fall generically into categories of usage including measuring plans vs actuals, technical quality management, technical performance management, coverage of product/test risk reduction.

Users are usually provided with a framework which hierarchically maps high-level behavioral goals to lower-level product features. One example of a framework is the *Extended RL Software Quality Framework* discussed previously.

There are both internal and external drivers of metric usage. In many cases, developers are required by contract to have funded measurement programs. Contractors may feel it necessary for the competitive edge to establish internally such programs. Too often, however, the external drivers forces result in just another burdensome document, generated by a separate group, for the use of only the customer for review. For those enlightened developers, who consider measurement an integral part of a mature software engineering discipline, the benefits of a measurement program are effective management through a better understanding of the project products, methods, and process. Measurement provides these developers with an opportunity to improve performance based on available past performance.

3.2.3.1 Costs

Several costs must be considered when implementing software product metrics. These include:

- *Metrics Utilization Costs* - Associated with each metric are the costs of collecting data, automating the metric calculation (when possible), and analyzing, interpreting and reporting the results.

- *Software Development Process Change Costs* - A certain set of metrics may imply a change in the development process.

- *Organizational Structure Change Costs* - A certain set of metrics may imply a change in organizational structure used to produce the software products.

- *Special Equipment* - Hardware or software may have to be located, purchased, adapted, or developed to implement a set of metrics. Tools currently being used in the development of software quality frameworks and metric computations and analysis are: ACT, Logiscope, PC-Metric, Tcov (Unix Utility which measures test coverage), Software Analysis Workstation (CASE Tool by Cadre Technologies), ATVS, ASQS, AMS, and QUES.

- *Training* - Quality Assurance or development may need training in the use of metrics and data collection procedures. If the introduction of metrics has caused changes in the development process, the development team may need to be educated about the changes.

A recommended method to assess costs is to identify a set of metrics to collect, analyze and interpret the data based on an established quality framework. The measurement process, just established, must then be subject to one or more pilot projects to verify or improve the estimated costs associated with it.

3.2.3.2 Benefits

Current benefits in using software product metrics include:

- Higher quality is achieved for end products and subsequent product development. This is possible because required quality levels are specified early on and quantitatively.

- By improving product quality, development realizes an improvement in the process used to build software products.

- A greater emphasis on quality is achieved throughout the software life cycle.

- Improved management control over software process, product quality and product evolution.

- Increased customer and user satisfaction.

- The value of software product metrics increase as they are applied to earlier phases or activities of the product life cycle.

3.2.3.3 Problems

There are potential problems that may arise when using software product metrics. These include:

- *Interpretation* - Subjective judgement is required in scoring some metrics, making uniform interpretation difficult.
- *Accuracy of Data* - The people gathering the metric data from the same software product could collect the data differently. There is a need for a standardized way of collecting and assessing results.
- *Availability of Automated Measurement Tools* - Most tools are language dependent.
- *Validity of Automated Measurement Tools* - Metric tools are relatively new to industry.
- *Organizational* - Who should collect the data?
- *Staffing Problems* - Inexperienced personnel are developing the measures and performing the measurement. Also, software development has no say in what is collected. This results in resistance and little, if any, process improvement.
- *Scope* - Is it better to measure one entire phase of the product life cycle or measure small sections of each phase of the life cycle?
- *Education* - Once management receives metric results little is won with them due to the lack of understanding.

3.2.3.4 General Issues

General issues discussed with respect to the state-of-the-practice for software metrics are:

- Does management and development understand the primary objective of measurement, that is, to improve the software development process and establish control over the engineering process?
- Who should own the metrics, that is, be responsible for using and maintaining the metrics and data?
- Who should collect, analyze and interpret the metrics?
- At what point in product development should the metrics be collected?
- When should the metric results be fed back into the software process?

The panel agreed that each of the above issues needed to be resolved early in the product life cycle and that collecting data may be a multi-organizational" effort. Prior to testing the product, it may be Development's role to collect the metrics during the initial compile and then prior to a code review. It then may become a quality assurance activity once the product is handed off for evaluation.

Since establishing a metrics program can result in high costs, the organization(s) developing the program must consider that measurement be:

- achievable (i.e. the metrics being considered are in fact collectible);
- complementary (i.e. the results of one metric for a quality factor may be used to assess another quality factor);
- repeatable (i.e. the results one user receives would exactly match the results another user receives); and
- the metrics must certainly satisfy and address the established quality requirements and specifications.

Potential areas of improvement for software product metrics include establishing a universal or standard set of software product metrics, integrating software metrics with system and process metrics, and developing metrics for trustworthy and fault-tolerant applications.

3.2.4 State-of-the-Art

Research over the last 15 years has developed a wealth of product metrics. Historically, most metrics have analyzed the software code itself or products directly related to the code (i.e., faults and failures). Most efforts have been generally concerned with the development phase of the life cycle. We are currently beginning to see more interest, research, and the existence of some tools dealing with other life cycle phases, specifically design and maintenance. Isolated cases are also surfacing in developing metrics for the requirements/specification phase. The analogy is that if it costs $1 to correct a line of code in requirements, $10 in design, $100 in testing, and $1000 in the maintenance phase - tremendous cost savings can be achieved if effective metrics can be developed for early life cycle stages.

3.2.4.1 Approaches

Problems in assessing state-of-the-art are that most companies are not prone to divulge current metrics practice and research for fear of losing competitive advantage or not wanting to publicize failure rates. The state-of-the-art viewpoints expressed in this section comes from recent publications and from the government and industry experiences of the panel participants. Metrics practiced today as a discipline is ad hoc and sporadic and has not reached an institutionalized stage in most companies. There are a few exceptions to this. There currently appears to be more concern and interest in advancing the state-of-the-art for metrics in the European and Japanese communities than in the United States. The Japanese are employing the use of simple metrics and collecting large amounts of data for metric analysis.

3.2.4.2 Use Scenarios

Since most metrics research and use has concentrated on software code analysis, a large percentage of metrics analyze code. Table 2 shows state-of-the-art metrics that are currently being computed.

TYPE OF METRIC	PRODUCT(S) ANALYZED
Consistency	Design, Software (map to requirements)
Control Flow Complexity	Software
Cyclomatic Complexity	Software
Design Structure	Design
Est. Number of Remaining Faults	Software Faults
Expandability	Software, Maintenance Product
Fault Distribution Within Code	Software Faults
Information Flow	Software
Interface Complexity	Software
Modularity	Software
Readability	Requirements, Design, Documentation
Residual Fault Count	Software Faults
Software Maturity	Software, Maintenance Products
Software Release Readiness	Software, Maintenance Products
Software Science Measures	Software

Table 2. Products Currently Analyzed by Software Product Metrics

3.2.4.3 Experiences to Date

Several frameworks have been developed which serve as a guide for software measurement. Further development and use of these frameworks are seen as essential to furthering the state-of-the-art. These frameworks will not only provide consistency for baselining the state-of-the-practice, but will also provide a basis for "within project" decision support throughout the life cycle.

• *RL Software Quality Framework*

As mentioned previously, the Rome Laboratory has developed a methodology for collecting software product information. The methodology rests on the foundation that software can be described by a number of factors (such as maintainability and self-descriptiveness). Each factor is characterized by a set of lower-level of criteria, the basis of which is a collection mechanism for detailed software information.

• *The TAME (Tailoring A Measurement Environment) Project*

TAME, a project developed at the University of Maryland under partial sponsorship from NASA's Software Engineering Laboratory, is an example of an tool environment that facilitates the derivation of software measures. By using TAME, software developers and managers can systematically define goals, choose the most appropriate methods and tools, and obtain quantitative feedback.

- *SEI Assessment Methodology*

  Although the SEI organization assessment methodology primarily addresses organizational process issues, by determining the degree of software engineering process maturity an organization possesses, the organization can determine the feasibility of implementing a program of product measurement.

- *Objectives Principles Attributes (OPA) Framework*

  The OPA Framework is similar to the RL framework. It starts with a set of top-level measurements and derives a mapping of these top-level measurements to a low-level set of metrics, or indicators. Examples of OPA indicators are cohesion, coupling, and ease of change.

3.2.4.4 General Issues

Although there are a large number of product metrics that have been proposed and researched, not many are being utilized by industry. Some reasons why industry has been reluctant to adapt product metrics are:

- Software engineering is not a true engineering discipline.

- Lack of general agreement on goals and functions of software quality assurance.

Additionally, while there has been recent work to develop metrics that assess all life cycle products, most metrics are still related to code and its associated attributes (faults and failures). One reason for this is the lack of uniformity and lack of formalization in the other products.

Other obstacles to advancing the state-of-the-art include:

- Not all metrics are suitable for all application domains. Work needs to be done on tailoring metrics to specific application domains or overcoming application differences.

- The lack of software engineering process maturity has hindered the application and usefulness of many metrics.

- Particular attention should be paid to who is using/interpreting the metrics. Metrics should convey the maximum possible information to the target audience, be it users, managers, or technical staff.

- Most state-of-the-art metrics have been tested in small-scale, usually unrepresentative settings (such as a educational environment or a particularly narrow applications environment). As a result, it is difficult to apply the metric to a large-scale environment and to estimate the cost of implementing the metric measurement program.

- Lack of artifacts (products of the software engineering life cycle) collected from a variety of sources reduces the possibility of validating a product metric.

- Ownership of metrics may affect: 1) the integrity of data collected 2) willingness to collect the data 3) thoroughness (preciseness) of data collected.

3.2.4.5 Known Future Directions

Future areas of product metrics research focuses on three general activities.

- Extending product metrics to measure all life cycle phases.
- Using metrics as predictors of the quality of successive life cycle products and for making product release decisions.
- Using metrics to evaluate different software engineering strategies, tools, and techniques.

One product that has received a great deal of attention pertaining to development of product metrics is the design product. The development of such metrics is related to more formal specification of the design. Aside from readability/comprehensibility metrics (if the design consists of English text), metrics for design include complexity and other types of structural measures. Some attention has also been paid to developing measures for products in the maintenance phase of the software engineering life cycle.

The second research direction concentrates on the use of product metrics as predictors of the quality of the to-be-developed products. An example is the use of a design metric to predict the complexity of source code [9]. In addition, Rombach has discussed applying product metrics for predictors of software in the maintenance phase [10].

The third research direction deals with using metrics to evaluate different software engineering strategies, tools, and techniques. This use involves gauging how effective our software engineering practices are and can offer insight for improvement.

3.2.5 Recommendations

To further the development and use of software metrics for maturing mission critical software technology, the following courses of action are recommended for software product metrics.

Recommendation 1. Classify Software Metrics and Define Desired Properties of Software Metrics

There are several metrics available in practice. However, metrics serve different purposes and users. Experiences with using metrics suggests a need for a standard classification scheme which contains information about objectives for use, such as, who is the audience (e.g., technical, management, user, owner, acquirer) and what is the purpose or desirable behavior (e.g., reliability, portability, maintainability). There are already several classification schemes in industry [11,12]. What is needed is a standard classification scheme for software product metrics. The classification will provide for consistency in understanding what the metric does and its purpose. Metrics should provide some benefits for being used. Defining what properties a product metric should have is essential for acceptance and adoption into practice. Metric definition and usage should mean the same thing within projects, companies and countries. Data propriety is a major concern but with sanitizing tools, project and company identity can be hidden. This use of sanitized data is essential to calibrating and validating software product metrics.

Recommendation 2. Conduct Research in Early Life Cycle Metrics

As electronic capture of requirements and design information becomes more widely used, the application of early life cycle metrics become easier. Using these metrics for predicting software quality factors and assessing compliance with system requirements will enable early identification of areas of non-compliance. A state-of-the-art trend is to focus attention on these early requirements and design metrics. Although current projects are addressing the computation and validation of early life cycle metrics for predicting and validating reliability and maintainability, additional research and trial application needs to be conducted prior to their being adopted for widespread use. For example, metrics for assuring that the requirements specification is complete, consistent, verifiable, and stable enough for release are needed. Additionally, the development and evaluation of design metrics for predicting operational reliability, maintainability, and supportability are also needed.

Recommendation 3. Develop Metrics for Characterizing Usage

The observed reliability of mission-critical software during operation is a function of its usage. Specification of this usage during the requirements phase is critical for early prediction of operational software reliability. Deviations from the specified usage and that observed can contribute to mission-critical software failure during operation. Understanding of the usage at all life cycle phases will enable the tester/analyst to be more effective during the conduct of life cycle-verification and validation activities. Deviations from usage before and after release, and from one operational release to another must be factored into certification and re-certification decisions. Considering the usage as a software product and developing this usage at all life cycle phases is a recommended research activity.

Recommendation 4. Conduct Research in Product Assessment Metrics

In the metrics world there are several metrics that support estimation and prediction of cost, schedule and product quality. As metrics are used more, there must be a way to validate and refine the metrics. Research is needed in augmenting assessment metrics. Assessment metrics will provide insight on how well the product behaves in the operational mode. Assessment metrics also indicate how valid and useful are the estimation and prediction metrics. Assessment metrics provide the necessary feedback and closure on the merits of the metrics used earlier in the development.

Recommendation 5. Conduct Research Integrating Product, Process, System and Acquisition Metrics

A need exists for a unifying view of metrics. Research should be conducted with the goal of establishing a framework which integrates the viewpoints of the system engineering, product metric and process metric communities, and the acquisition managers.

Recommendation 6. Further Refine, Develop, and Integrate Tools to Support Product Measurement of Languages and Architectures

In order for the use of metrics to become cost-effective and widely accepted, tools and metric tools need to be developed. Future tools should be instrumented for collecting metric data. One tool that has been developed by RL, with this concept in mind is the Ada Test & Verification System (ATVS). ATVS is a testing tool that collects quality data from detailed design through testing phases. Software development environments (e.g., Software Life Cycle Support Environment) can couple either loosely or tightly, tools to collect and store metric data into the project database for supporting subsequent metric tool evaluations. The use of tools and environments would allow subsequent validation and promotion of metrics. Quality Evaluation System (QUES), a metric tool being developed by Rome Laboratory, will support not only the RL Software Quality Framework but the AFSC Software Management and Software Quality Indicators. QUES integrated into SLCSE can gather metric data from the project database and provide quality evaluations at phase of the software development life cycle. QUES can interface with various development tools to collect metric data and provide necessary metric reports on the quality of the products. QUES will increase automation of data collection and decrease the cost to collect metric data. QUES will also be capable of supporting any type of metric scheme. Quality editors should be developed to provide real-time feedback on the quality of the specifications, design and code. If thresholds are violated, the user is notified that there is a problem. It then will be up to the user to make the necessary corrections to insure that the product is at the required quality level.

Metric tools will have to support procedural, declarative and visual type languages. Various architectures (e.g., parallel, distributed) will also need to be supported by tools and metrics in order to capture the quality of the products. In addition, metrics for safety, security and trustworthiness need to be defined and developed.

Recommendation 7. Conduct Research in Metrics for Trustworthy and Other High Integrity Applications

Research should be conducted in metrics for special factors[1] such as nuclear safety, computer security, and flight safety. Some of these special factors are related to specific application areas, like computer security and flight safety; others are more general, like trustworthiness. The latter is of special interest to the Technical Panel on Trustworthy Computing Technologies (XTP-1). The research should seek to clearly define these factors, relate them to existing criteria or propose new criteria, and investigate metrics that are applicable.

The relationship between the formal methods used to develop demonstrably trustworthy computer systems and software metrics considered in this workshop is not clear. Research is needed to investigate this relationship and to examine in detail those areas where formal methods and software metrics can complement each other. Some initial questions are:

- To what extent do formal specifications influence the application of design metrics?
- Assuming that only part of a software system can be proved correct in accordance with its specification, how can software metrics be used to increase the overall trustworthiness of the system?
- Assuming that all of a software system can be proved correct, what software metrics are useful for increasing the overall trustworthiness of the software?

[1]Use of this term is based on the RL *Software Quality Goal Framework* of factors→criteria→metrics.

- Are there useful metrics that can be applied to the results of automated proof?

Recommendation 8. Develop a Mechanism for Technology Transfer

In order for metrics to become widely used and accepted, a National Initiative should be established. This initiative could build on the ideas proposed in [13]. For example, it could pull together work done in metrics by establishing a Software Quality Laboratory (SQL) with its sole purpose to transfer metric technology to the world. The SQL would be a vehicle for training how to use metrics, collect data for validation and calibration, provide analysis and feedback, configuration management of the metrics for state-of-the-practice and research for development of new metrics or deletion of non-contributing metrics. The SQL would also be used for correlating metrics with factors and good software engineering practices. Under this initiative a vehicle would be in place to transition metrics to software developments. The data collected would be used to further refine and calibrate the metrics.

As systems become more complex and software intensive the more difficult it becomes in controlling and managing the development. Having effective metrics in place will allow managers and developers to have a grasp on the quality of the products in the early life cycle phases. These product metrics also provide a check on the process and if the product is below quality standards then there is a problem with the process that produced the product.

A data repository should be established that provides the lessons learned in application of product metrics. What is lacking in the world of metrics is a way to "spread the word" on what metrics are good, how to use the metrics, how to present the information, and finally how to interpret the information once it is collected. Getting metrics into practice is difficult without commitment. But if there were document cases with data of success cases, metrics would be more readily accepted.

Recommendation 9. Develop Mechanisms for Effective Reporting of Software Product Quality Metrics

A major stumbling block in getting metrics used and accepted is the lack of reporting vehicles. How and what information do you present to management, a developer or a maintainer that will portray the information in a concise and meaningful way? For a manager, measures of graph complexity have no meaning, but the fact that the software will be difficult to maintain and unreliable is what the manager wants to know about the product. Research needs to look at how to present and interpret the information for the various users of the metrics. Once the users are capable of understanding the metrics, they are in a position to be able to grasp the problem at hand and make the necessary decisions. Guidelines for reporting the information to various users should be developed.

Recommendation 10. Address Cost/Benefits of Using Metrics

What metrics are being used and the extent of their use varies widely among industry practice. Further refinement and assessment of metrics will be limited by the degree with which they are being used by mission-critical software developers. The cost-benefits of using metrics will be a significant factor in industry acceptance and use. In the past attempts to lower life-cycle costs have focused on productivity improvement, for example, through automation of processes. What we have learned from total quality control advocates is that productivity is a byproduct of a quality improvement program. Productivity is increased due to problems being detected earlier thus requiring less rework.

This recommendation calls for research efforts to examine the distribution of cost, improvements in product quality and reductions in costs due to the application of software product quality metrics. Publications documenting lessons learned, metric data, and cost/benefits experience will provide useful examples for industry.

3.2.6 References

[1] T.J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, S-2, Issue 4:308, December 1976.

[2] S. Henry and D. Kafura, "Software structure metrics based on information flow", *IEEE Transactions on Software Engineering*, 510-518, September 1981.

[3] J. P. Cavano, "Toward high confidence software", *IEEE Transactions on Software Engineering, SE 2,* Issue 4:1449-1455, December 1985.

[4] M. Sheppard and B. Ince, "Metrics, Outlier Analysis and the Software Design Process,", *Information and Software Technology,* 31, Issue 2:9198, March 1989.

[5] M. Halstead, *Elements of Software Science.*

[6] Rome Laboratory, *Specification of Software Quality Attributes.* Technical Report, RADC-TR-85-37, Griffiss Air Force Base, Rome, NY, 1985.

[7] Air Force Systems Command. *Software Quality Indicators.* Technical Report AFSC Pamphlet 800-14, Department of the Air Force, Andrews Air Force Base, 1987.

[8] Watts S. Humphrey, Managing the Software Process., Addison Wesley, 1989.

[9] Sally Henry and Calvin Selig, "Predicting Source Code Complexity at the Design Stage", *IEEE Software,* March 1990.

[10] H. D. Rombach and B. T. Ulery, "Improving Software Maintenance Through Measurement", *Proceedings of the IEEE,* April 1989.

[11] IEEE standard dictionary of measures to produce reliable software, 1989.

[12] IEEE guide for the use of standard measures to produce reliable software, 1989.

[13] Edward R. Comer and Andrew J. Chruscicki, *Concept for Establishing a National Initiative for Software Quality Improvement and a Recommended Role and Plan of Action for the Rome Air Development Center,* Technical Report, Rome Laboratory, April 1990.

3.3 Panel 2: Software Process Metrics

3.3.1 General Information

3.3.1.1 Panel Participants

NAME	EMPLOYER	COUNTRY
William Agresti, Chair	MITRE Corp.	USA
Joseph Batz	OUSDA (R&AT)	USA
Robert Cruickshank	Software Productivity Consortium	USA
Ray Engelman	TRW	USA
Stewart Fenick	US Army/CECOM	USA
Kenneth Littlejohn	US Air Force/Wright Laboratory	USA
Rhoda Novak	Aerospace Corp.	USA
Robert Park	Software Engineering Institute	USA
Richard Selby	University of California/Irving	USA
C.K.S. Chong Hok Yuen	College Militaire Royal de Saint-Jean	Canada

3.3.2 Introduction

The Process Metrics Panel, in setting the stage for its discussions, agreed to begin with the definition of process proposed by Watts Humphrey in his book, Managing the Software Process [1]: Process is the set of activities, methods, and practices that are used in the production and evolution of software.

Humphrey elaborates on this definition by observing that there are many processes within any software development, and there are several levels at which processes can be identified and measured.

In the sections that follow, we endorse and reemphasize these observations. We do this through listing and illustrating a number of scenarios, experiences, and general issues that relate to making software process metrics productive. In each instance, we find that the underlying and primary concern is to have defined processes to be measured. The first requisite for process metrics, then, is to construct definitions (or models) of the processes of interest.

Process metrics, as we view them, may be at the "big-P" (project) level, the "little-p" (atomic level), or anywhere in between. Big-P process metrics are those that program managers and acquisition agencies can use to plan, monitor, and control activities, resources, and schedules at the contract or project level. These are the most prevalent metrics and the ones with which managers have the most experience.

Little-p process metrics support individuals and first line managers controlling and improving the local processes under their own immediate supervision. If the history of hardware process improvement is a guide, persistence in the application of little-p metrics may prove to have impacts that exceed those experienced to date with higher level project measurements. Little-p metrics are essential to the process improvement paradigm that underlies total quality management (TQM).

In order to sharpen its focus on process, the panel attempted to identify some principal differences between product and process measurements. These differences take one of two forms:

- Process metrics addressing effort or resources.
- Process metrics introducing a time dimension.

Examples of effort and resource metrics include staff hours, staff levels, productivity, staff experience, capital investment, and training. Time-associated examples include elapsed intervals for activities (e.g., schedules), time derivatives (rates), profiles, and the occurrence of events. The latter includes starting and stopping of activities or phases, insertion or removal of errors, and the arrival or identification of changes and change requests.

The panel also observed that any product measure can be transformed into a process measure through the way it is applied. For instance, by measuring a product at more than one point in time, rates, slopes, trends, and profiles can be derived. Thus, product measures can be viewed as contributing to process measures.

Another point mentioned was that, by being formal when codifying a process, one can legitimately take the view that processes are themselves products. This view was first advanced by Lee Osterweil in his plenary presentation at the 1987 International Conference on Software Engineering in Monterey, California entitled "Software Processes Are Software,Too" [2].

The final stage-setting observation was that measures always depend on the process method being followed. This implies that it may be precipitous to attempt to legislate specific details of process metrics, at least for lower level activities, until the local issues of process definition have been settled. Often the simple identification and description of specific software processes suggest immediate and natural points at which useful quantitative observations can be made.

3.3.3 State-of-the-Practice

The panel decided to organize its considerations related to assessing the state-of-the-practice of process metrics around the following seven key concerns.

3.3.3.1 What process measures are being used in local or global settings?

Process measures currently being used in local or global settings were reviewed by the panel. Twelve process measures currently in use were identified as representative examples of state-of-the-practice process measures. These measures included:

- Human Resources Consumption/Effort Measures - the number of staff hours consumed for a particular activity.

- Computing Resources Measures - computer resources used in development, including memory/network/CPU (connect hours, CPU seconds) measures.

- Errors/Defects Measures - the number, severity, and type of errors found during development and maintenance.

- Change Measures - the number, severity, and type of changes initiated both internally and externally during development and maintenance.

- Schedule Measures - the duration, and start/end times of schedule-related activities.

- Cost Measures - the costs incurred in labor months and dollars from activities performed.

- Capital Resources Measures - the dollars expended for training, hardware, and software purchases.

- Trends Measures - variances between actual and planned measures of schedule, resource consumption, etc.

- Productivity Measures - source lines of code (SLOC) or function points developed in relation to unit of effort expended. (Note: The current draft of the IEEE Computer Society Standard for Productivity Measures contains more discussion of productivity measurement issues and alternatives.)

- Personnel Measures - the levels of staff experience with application area, development environment, etc.; numbers of designated key personnel; staff turnover.

- Conformance to Standards/Practices Measures - the extent to which the processes and activities are conforming to defined standards and practices.

- Code Growth Measures - the change in code size (under configuration control in the project library) over time.

A wide range of additional process metrics is used throughout industry, but the extent of use varies. Consistently obtaining estimates of the process metrics above enables the comparison of actual values to estimated or planned values. The code growth measure is an example of the case mentioned in the introduction: a product measured at regular intervals serving as a process metric by providing visibility into the process that results in code entering controlled libraries. An array of similar progress metrics offers visibility into the process. Examples are the rate at which computer software units (CSU's) are coded, reviewed, and unit tested.

3.3.3.2 What is industry's experiences with the cost of process metrics programs?

The panel members were concerned about citing a particular percentage of total software development costs that organizations could expect to devote to process metrics data definition, data collection, and reporting. Instead of citing a cost figure which may be misinterpreted, the panel recommends references [3, 4, 5] which discuss the cost of metrics programs.

A few general cost issues were discussed by the panel. For example, the costs certainly vary depending on the scope of the program and the degree of automated support available. Certain Big-P metrics are readily available--for example, the size of a contract in dollars and the budget of a software development department. Taking advantage of existing configuration management tools can simplify obtaining the code growth metric. Similarly, labor cost data that is extracted automatically from the work breakdown structure (WBS) database will lessen the cost of data collection. When labor data must be extracted from the WBS by human effort, then it is semi-automatic. When labor data is collected by examination of time cards or other individual labor-recording forms, then it is human-intensive and more expensive to collect.

Standard definitions of both recurring and non-recurring costs in software development and maintenance must be established. These definitions can only be expressed in terms of the activities involved in the process. The expertise of financial as well as software development personnel is needed, and both groups must be cross-trained in these disciplines.

Costs will be attributable both to individual projects and to the organization as a whole [3]. Effort will be expended in designing forms and in the actual collection of data but the costs of database maintenance and management reporting should also be considered. Experience shows that process metric data, if properly collected, analyzed, and organized, will more than pay for itself with more effective management of future projects [3].

3.3.3.3 What problems are encountered in process metric programs?

Various problems are associated with process metrics in common use. We have identified several problems and grouped them into seven classes as follows:

3.3.3.3.1 No sound theoretical measurement methodologies exist

One of the main problems concerning current process metrics is that they are mainly of an *ad hoc* nature. The panel believes it would be better to start by describing the process, establishing goals to accomplish, and identifying attributes to measure. Then, it will be easier to define and select metrics that will have an understood relationship to specific goals. Unfortunately, there are at present, no sound and comprehensive measurement methodologies, no standard definitions, and no standard metrics.

Another problem seems to be that many of the properties which we attempt to measure (e.g., quality and complexity) are inherently difficult to quantify. Existing metrics for these properties are not completely satisfactory, measuring only some particular aspect of each property. Very often, too, these metrics are very weak measures, of the nominal or ordinal type on the measurement scale.

3.3.3.3.2 Data collection

Another problem associated with process metrics, and metrics in general, is that it is often cumbersome to collect the data, even if we know what metrics to apply. Data collection is often perceived to be expensive, difficult and not very useful, although this may be a wrong perception. The data collected may not be sufficiently representative and it may not be collected in a consistent fashion "in space" (e.g., the contractor and the different subcontractors may not collect the data for the same metric in similar ways) and over time by the same organization.

3.3.3.3.3 Data Analysis, Interpretation and Validation

Once the data has been collected, it has to be analyzed, interpreted and validated. Software engineers, generally speaking, do not have the necessary statistical sophistication for performing these tasks. Very often, the results are misinterpreted, inconsistently applied, or, simply, misapplied. Very often, too, there is no feedback of the results so that no benefits are derived from the measurements, and corrections are not made to improve the process.

3.3.3.3.4 Tools

Very few useful automated tools exist for data collection, analysis, validation and interpretation. Generally speaking, when they do exist, the tools, are immature, not integrated, or both.

3.3.3.3.5 Management

Management understanding and support of metrics and metrics-related efforts was characterized by the panel as being marginal at best. Experience shows that higher level management will give verbal and/or substantive support to metrics efforts because it is a "good thing" for management to measure process and products. But metrics efforts are more likely to receive inadequate support from middle management because, perhaps, it is this group that has budget responsibility. There appears to be a lack of understanding of the structure and purpose of metrics. There is a perceived high cost of entry into a metrics program, and it is also perceived that the payoff horizon is too long (the "not on my watch" syndrome). In summary, the benefits are not obvious.

3.3.3.3.6 User Resistance

There is resistance to the use of metrics based on a *fear of audit*. The fear is that metrics, if the numbers are deemed unsatisfactory, will be used as a tool to punish through bad evaluation reports. It is important for these perceptions to be dispelled.

3.3.3.3.7 Institutional Resistance

Resistance to using software metrics also exists because there are no perceived success stories. This attitude is based on the perception that "my project is unique". In truth, there is usually enough commonality among projects for metrics to be equally applicable and useful while still retaining project specificity.

Institutions often don't appreciate that they can achieve long term benefits with a process metrics program that includes multiple projects over time. The metrics help establish a useful quantitative baseline. Also, the centralization of a metrics program in an institution may provide some efficiency through economies of scale.

Changing project environments often discourage projects or organizations from metric usage. But, ironically, metrics can be helpful by detecting these changing environments.

Finally, institutions are concerned that too much project data and metrics visibility will result in proprietary data being involuntarily released to the outside world. Proper security procedures need to be developed to address this concern.

3.3.3.4. What are the benefits of process metrics programs?

This section attempts to identify the benefits which have emerged or have been identified in software development projects as a result of an implemented software process metrics program. The panel identified the benefits of process metrics as follows:

a. Metrics provide visibility into the process: as a minimum, an awareness of what's going on; as a maximum, insight into why.

b. Metrics are used to assess conformance to project goals and software requirements.

c. Use of metrics information (status, progress, problem pointers, trend indicators) leads to formal/organized project control.

d. Control can mean maintaining the status quo, or dictating corrective action - i.e., metrics information helps management direction or redirection.

e. Metrics information is used for planning purposes - future strategies, next project considerations, hiring and staffing plans.

f. Metrics facilitate feedback and warning activities and, if done properly, these activities will occur early-on in the software process.

g. Metrics information is used as input to the process of determining award fees.

h. Metrics information is used at the little-p level to motivate/implement "improvement programs/strategies" by the responsible people and departments.

i. Metrics are used to improve an organization's near-term performance and long-term capability through lessons learned.

j. Metrics can be used to aid the vendor selection process, and then, to assess on-the-job performance to see if the selected vendor is living up to the "contracted capability/maturity level".

k. The ultimate "bottom-line" benefits are improvements to the software process (cost, schedule, productivity, capability) and to the software products (quality, performance, functionality, usability, customer satisfaction).

3.3.3.5 Why are process metrics being used?

Metrics are being used to :

a. Realize the benefits delineated in Section 3.3.3.4.

b. Assess the fulfillment of mandates from government acquiring agencies.

c. Assess the fulfillment of organizational internal mandates.

d. Acquire lessons-learned for cross-project and next generation software improvements.

e. Help assure customer satisfaction with the final result (product, cost, timely delivery).

f. Help bring large, complex software systems under manageable control.

g. Reduce the risk of:

(1) having to do extensive rework

(2) settling for a modified version of the original requirements

(3) cost overruns

(4) slipped schedules

(5) paying many times more for *testing errors out* rather than *building quality in.*

3.3.3.6 What are examples of state-of-the-practice use?

The panel identified and discussed several representative examples of state-of-the-practice use, especially in DoD-supported efforts. The U.S. Army Missile Command (MICOM) has developed Software Development Status Indicators and are using them currently on MICOM system developments. This methodology makes use of actual versus planned graphs and red, green, amber warning flags to the Project Office.

The MITRE Corporation and the Air Force System Command Electronics Systems Division (AFSC/ESD) developed the original software management indicator set [*Software Reporting Metrics (1985)*] which has since undergone additional development and refinement [*Software Management Metrics (1988)*].

AFSC leveraged off the work of their ESD organization and produced the Software Management Indicators and Software Quality Indicators Pamphlets AFSCP 800-43 (1990) and 800-14 (1987). A draft revision has been released which combines both Pamphlets into one and also adds an addendum draft Data Item Description (DID).

AMC (Army Material Command) has adapted the AFSC pamphlets as AMC-P 70-13 (1987) and 70-14 (1987).

A Software Metrics Analysis & Reporting Tool (SMART) was developed for Project Manager Field Artillery Tactical Data System (PM FATDS)'s Advanced Field Artillery Tactical Data System (AFATDS) development. The tool operates on a slightly modified set of the AMC (also AFSC) Software Management Indicators called the Software Management and Quality Indicators (SMQI). An SMQI implementation guideline has been developed.

Naval Underwater Systems Center (NUSC) has developed a start-to-finish metrics program and is using it on Navy software projects.

The panel reviewed classes of tools used for process metrics as follows:

• Line Counters - These tools are used today in many organizations to automate the collection of statistics on source code composition and size. Although such tools are prevalent, we noted that most automated counters have not been built through the same industrial strength processes that are normally used to develop commercial products. Code may go in and numbers may come out, but the rules used to arrive at the numbers are often either unknown or unvalidated.

• Test Tools - Several tools collect source code testing-coverage metrics. Example metrics are percentage of statements, branches, basis paths, and paths executed.

• Source Code Analyzers - Many commercial tools analyze source code and produce reports that profile the code by statement type, etc. and provide various complexity measures on the code. An analyzer in the public domain is the Static Source Code Analyzer Program (SAP) for FORTRAN [6].

• Cost Estimation Models - Many commercially available packages implement cost estimation models. The models estimate development cost based on factors such as estimated size and characteristics of the development organization.

Another characteristic of a state-of-the-practice organization is a metrics database spanning multiple projects.

3.3.3.7 What are general issues associated with state-of-the-practice process metrics?

The following is a list of general issues addressed by the panel based on the current state-of-the-practice in the area of process metrics. This is not a comprehensive list of all issues, but a representative list of concerns which this panel addressed.

3.3.3.7.1 Big-P versus Little-p (Macro-Level Process versus Subprocesses)

At what level should metrics be applied to produce the most benefit? It was the view of this panel that a macro-level process (big-P) is made up of potentially several subprocesses (little-p) and that a process is measured in part by measuring intermediate products produced by a process. Therefore, the concern is whether metrics that focus on the entire life-cycle software development process (big-P) would produce the more useful information for future development efforts or would a view of the individual processes (design process, development, etc.) yield the more useful information. One of the purposes of applying metrics is so that one is able to better understand the software development process and analyze the strong and weak points of an effort to improve the development process. The panel recommended that a distinction be made between the two levels when applying metrics.

3.3.3.7.2 Applying Measurement Technology to Software

The panel identified several issues related to the application of measurement technology to the analysis of software. Since the technology is in an infant stage, and there is no particular science defined, nor a methodology that outlines measurement technology, the following issues need further definition:

3.3.3.7.2.1 The scales used for metrics.

In the application of a metric, the scale needs to be consistent with its use at selected points in the life-cycle of the development process. The scale used should also be clearly defined, be relevant to the project, and defined at a level such that the results are easily understood by the user.

3.3.3.7.2.2 The terminology used in reference to metrics.

Ambiguity exists in the terms used in the definition of metrics. For example, does the definition of a source line of code in Ada refer to a count of semicolons, carriage returns, executable lines, or non-comment, non-blank source statements? This is just one example of how terms are not clearly defined and the misinterpretation can lead to ambiguous and unclear results.

3.3.3.7.2.3 The proliferation of metrics and the inherent problem of overlapping metrics.

Inherent to the problem of the proliferation of software metrics are the issues of interrelationships among metrics and different metrics covering the same information. The information each metric covers should be clearly defined in an effort to prevent "the use of metrics for metrics sake".

3.3.3.7.2.4 The abstraction level of metrics.

The panel addressed the difficulty of interpreting a metric due to its corresponding level of abstraction. Since measurements are abstractions of products and processes, metrics are inherently abstract, implying information is lost. This being true, the resulting information provided by the metric is unclear and is potentially not useful.

3.3.3.7.2.5 Reuse of Metrics.

Metrics should be defined to be useful across projects.

3.3.3.7.2.6 Impact of the Development Process Model

This issue was addressed because the selection of a particular development process model would call for the use of a particular set of software metrics. The ideal situation would be that a particular software metric would be applicable to any of a number of development process models, but this is generally not the case. The metrics implications play a role in the selection of a particular development model to use on a development effort. One would prefer a development process that has a mature metrics set associated with it.

3.3.3.7.2.7 Validation of Metrics

The issue of validation was raised by this panel in an effort to point out that the validation of metrics must depend on sound statistical techniques. The need exists for a means of determining the authenticity of the measurement taken, and this must be based on proven and acceptable techniques. Of particular interest to this panel are normalization, sampling, and inference of the measurements.

3.3.3.7.2.8 Contractual Issues

If metrics are called out in a contract, then data rights are an issue. Does the issuing body of the contract have unlimited rights to all metrics used on the contract, or only to those that are not deemed "proprietary"? The issue is to clarify the information to which the government has rights.

3.3.4 State-of-the-Art

The panel decided to organize its considerations related to assessing the state-of-the-art of process metrics around the following four key dimensions.

3.3.4.1 What approaches characterize state-of-the-art process metrics programs?

The panel outlined some of the methods and techniques that characterize state-of-the-art metrics programs. State-of-the-art metrics programs take maximum advantage of existing tools that are or can be instrumented to collect and analyze software metric data. Common examples are configuration management (CM) tools and compilers. A CM tool could be instrumented to collect changes in intermediate products, such as design documents, source code, etc. Compilers could be instrumented to collect source code metrics and to do data flow or other analyses. The collected metric data could automatically be stored in a shared database. Incorporating collection and analysis into these tools helps institutionalize metric use and lower entry barriers.

Knowledge-based tools or expert systems tools can provide assistance to developers throughout the development and maintenance processes. The rules in the knowledge-based tools could be determined from metric data derived from intermediate products and processes. Such a tool monitors metric values: when some combination of metric values is within a specified range, remedial plans are suggested. An example system under development is the Software Management Environment (SME) at NASA/GSFC [7].

If developers are able to fully codify a software process in a process language, then we could collect metrics based on this explicit process representation. We could also incorporate measurement-based feedback mechanisms directly into these processes. Languages that have been used to represent processes include several commercial languages and APPL/A at the University of California at Irvine [8].

The SEI process maturity framework has proposed that software measurement be integrated with process representations to create *empirically guided processes*. These processes have level-5 process maturity and are also called self-adaptive processes. Their key feature is that measurement-based feedback mechanisms are used to guide the processes while the processes are still being executed, as opposed to providing benefit only to subsequent processes. An example system under development that will support level-5 processes is the Amadeus system at the University of California at Irvine [8, 11].

A capability has been developed to query DIANA (Descriptive Intermediate Attributed Notation for Ada) representations of Ada programs and to return metric values. This query capability helps progress toward standardized metric definitions [9].

A general state-of-the-art trend is the integration of metric definition, collection, analysis, and interpretation/feedback tools with software development environments. Two examples of software environment projects that include software metric tools are Arcadia [8, 11] and TAME [10]. The Arcadia environment includes the Amadeus system mentioned earlier.

3.3.4.2 What are state-of-the-art use scenarios for process metrics?

One scenario for use of state-of-the-art process metrics is integrated data collection and reporting across industry and government. Some of the benefits would be consistent metric definition and interpretation, baselines for model calibration and evaluation, and increased sharing of knowledge. Of course, some aggregation of the data would be needed to protect company interests, anonymity of individuals, and any proprietary data. In such a scenario, metric data might be made available electronically to the government as a contract deliverable.

Knowledge-based systems and other database-centered tools can be designed to have extensible repositories of information. Newly collected metric data could be automatically incorporated into these systems. There might be two separate databases, one for a current project and one for general lessons learned across multiple projects. Tailoring tools could help calibrate the predictions to individual circumstances. These databases could be used internally in an organization to provide a self-assessment vehicle or to guide future projects. As mentioned above, parts or aggregations of metrics databases may also be made available to clients, IV&V analysts, or multi-organizational data repositories. An IV&V contractor may use some of the metric data in an analysis of the project.

3.3.4.3 What are experiences to date with state-of-the-art process metrics?

The panel discussed several metrics projects that are in very different stages of research and development.

COQUAMO (COnstructive QUAlity MOdel) is an activity of the European Strategic Programme for Research in Information Technology (ESPRIT) Project to estimate project quality using a parametric model in the spirit of COCOMO, a widely used cost model.

Rome Laboratory (RL) has developed a Software Quality Framework (SQF) and a number of tools/environments to implement the framework. These include the Automated Measurement System (AMS), a data collection and analysis tool designed to automate evaluation phase of the SQF, the Quality Evaluation System (QUES), a second generation data collection and analysis tool, the Assistant for Specifying the Quality of Software (ASQS), an expert system which provides support for the quality specification phase of the SQF, and Software Life-Cycle Support Environment (SLCSE), an integrated tool set which builds on many previous efforts.

Expert System for Software Design Analysis (ESSDA) is a Small Business Innovative Research (SBIR) project (Phase II) of the U.S. Army Communications-Electronics Command (CECOM) Product Assurance and Test (PA&T) Directorate which is looking at automating metrics and indicators for the design activity.

The U.S. Army CECOM Center for Software Engineering (CSE) is developing a CECOM Mission Software Management Metrics Methodology of tailorable metrics sets/subsets and metrics application techniques based on mission profiles. CSE is conducting a Users Experiences Survey and the resulting lessons learned will contribute to the methodology development.

The Technology for Assessment of Software Quality (TASQ) is an integrated quality measurement environment developed by U.S. Army Armaments, Munitions and Chemical Command (AMCCOM) and now under management of U.S. Army CECOM for revision.

There are many locally defined metrics programs. Defense industry contractors are realizing the benefits of metrics and are initiating internal programs for self-improvement purposes. There are a number of innovative Internal Research and Development (IR&D) projects developing metrics and metrics methodologies.

The panel identified SME, Amadeus, and TAME as instances of a metrics capability being integrated into the design of the environment.

The Software Management Environment (SME) is being developed in the Software Engineering Laboratory (SEL) at NASA Goddard Space Flight Center (GSFC). SME compares process metrics trends for an ongoing project to nominal trends from completed projects. A rule-based component of SME will suggest possible explanations for deviations from the nominal case [7].

Amadeus is a measurement and empirical analysis system integrated into the Arcadia environment. Amadeus supports SEI level 4 and level 5 processes and provides an extensible integration framework for metric definition, collection, analysis, and feedback tools. Amadeus is being developed at the University of California at Irvine under the direction of Rick Selby [8, 11].

TAME is a measurement system that supports the improvement paradigm. TAME provides capabilities for tailoring a measurement program to particular goals and projects. TAME is being developed at the University of Maryland under the direction of Victor Basili and Dieter Rombach [10].

Ginger is a UNIX-based tool that uses existing UNIX capabilities (such as wc and diff) to analyze software development activities and intermediate products. Ginger is being developed at Osaka University in Japan under the direction of Koji Torii.

Measures of intermediate products can provide insight into the effectiveness of the process that generated the product. For example, tools that analyze code indicate whether coding practices (part of the process) were followed. EASE (ESD Acquisition Support Environment) analyzes Ada systems by examining the intermediate representation of the system as DIANA-trees (Descriptive Intermediate Attributed Notation for Ada) [9].

Several organizations active in software metrics research, analysis, and collection include the NASA/GSFC Software Engineering Laboratory, the Software Productivity Consortium (SPC), RL, SEI, NUSC, CECOM, MITRE, AFSC/ESD, MICOM, and TTCP.

The Software Engineering Laboratory (SEL) is sponsored by NASA/GSFC and supported by the University of Maryland and Computer Sciences Corporation. The SEL has collected data on over sixty software development projects since 1975 and conducted experiments and analyses on software methods and tools [3].

The Software Engineering Institute of Carnegie Mellon University has three activities underway in its Software Measurement Initiative that are aimed at transitioning a portion of the state-of-the-metrics-art into metrics practice. These activities are:

- The Software Metrics Definition Working Group. This group, through its subgroup tasks, is structuring operational definitions of metrics for software size, quality, effort, and schedule. Its expressed purpose is to provide not just definitions, but a framework and formats for their operational application, and guidelines for use and interpretation

- The Software Acquisition Metrics Working Group. This working group is tasked with identifying and structuring a minimal set of metrics and practices that will help acquisition managers implement and use metrics for project/contract level planning and control.

- The Metrics in Practice Effort. This activity is seeking to capture, document, and publish success stories (and perhaps counter examples) of metrics use. Its purpose is to make available anecdotal evidence to support and motivate those who would like to establish or promote metrics use.

3.3.4.4 What are general issues associated with state-of-the-art process metrics?

A central issue in process metrics is how software processes are represented. An explicit process representation enables the formal analysis of a process itself, as opposed to the analysis of reflections of a process (such as the amount of effort consumed during the execution of a process).

Consistent metric definitions are useful because they enable comparison of metric data across environments and help the standardization of tools developed to collect the metrics.

Since software development techniques are continuing to evolve, we need methods to determine the useful lifetime of historical metric data. Baselines of metric data need to be recalculated periodically so that they remain representative of future projects and can be used to calibrate predictive tools.

Appropriate incentives for developers and managers need to be defined so that development personnel are motivated to use measurement techniques. A significant number of the obstacles for using measurement techniques are managerial and not technical. This is especially true for the state-of-the-practice approaches. As more and more projects use measurement approaches, there will be more sharing of data and lessons learned across projects and organizations.

Several metrics have been proposed to measure the various quality factors, sometimes called the "ilities". We need to determine metrics that quantify each of these quality factors and to build consensus around these metrics. These consensus-building activities would help enable the use of both process and product metrics as integral parts of software development and maintenance contracts.

Other general issues are the granularity at which data should be collected (e.g., unit, subsystem, system level), appropriate mechanisms to normalize metric data (e.g., by size, time), and the integration of multiple metrics and metric collection tools.

3.3.5 Recommendations

The Panel recommends five actions to advance the state of process metrics.

Recommendation 1. Develop a metrics maturity framework.

Software metrics are just now receiving acceptance as an important means of measuring and evaluating the processes and products of software development and evolution. However, much of what exists is still considered state-of-the-art and that which is considered state-of-the-practice must be applied very selectively. Furthermore, the ability of a development team to apply software metrics is largely dependent upon the professional software engineering maturity of its staff members and its management.

Therefore, this panel recommends that the Departments and Ministries of Defense of the TTCP member nations sponsor the development of a metrics maturity framework. The metrics maturity framework should:

- be structured as stages of increased maturity, as in the SEI Process Maturity Framework,
- be oriented to the iterative "real world" process as portrayed by the Spiral Life-Cycle Model or other evolutionary development processes,
- provide for metrics at each phase of activity in each iteration of the process,

- clarify the state-of-the-art versus the state-of-the-practice, and
- identify (quantify) benefits

Recommendation 2. Validate promising state-of-the-art metrics technology.

To close the gap between state-of-the-practice and state-of-the-art, TTCP member nations are encouraged to transition promising approaches. One mechanism is to define a selective "shadow" metrics activity that would, in parallel with a development project, try out a new metrics methodology or tool. A second mechanism is to support the validation of metrics methodology in a production setting. State-of-the-art approaches to metrics often need opportunities to address the scale-up to larger applications.

In addition it is important to obtain experience and accumulate data in the use of software metrics. Therefore, the panel recommends that software/systems programs be encouraged to:

- survey applicable and available metrics and techniques,
- define its own software process and subprocesses,
- collect data on the software products and processes,
- publicize success stories,
- perform analyses and evaluations of the impact of applying such metrics techniques and document these results, and
- use state-of-the-art metrics as "shadow metrics activities".

Recommendation 3. Promote software Work Breakdown Structures (WBS) which reflect the process defined by the developer.

Cost has always been a critical "management metric" but cost information is often submerged in "subsystems" costs in software-intensive programs. To facilitate the collection of cost data for process metrics, the panel recommends that:

- Software always be identified as separate (from hardware) configuration items.
- Software Configuration Items be directly traceable to the Work Breakdown Structure in order to clearly separate software costs from that of other systems elements.
- Work Breakdown Structures be tailored to represent the defined software process(es).

Recommendation 4. Promote process improvement and its integration with a metrics program.

The employment of software metrics strongly supports the objectives of total quality management (TQM). The ability of a team to improve its process and its products is a fundamental concept of TQM. A metrics program fulfills a central role in process improvement by providing a quantitative characterization of the process and recording improvement trends.

The SEI Software Process Maturity Framework is primarily a tool for self-evaluation, the first step in self-improvement. The highest level (level 5) of the Software Process Maturity Framework represents the highest level of software creation capability and therefore equates to the highest level of TQM for software. This level represents the ability of a team or organization to:

- repeat its level of control to meet commitments, costs, schedules and changes,
- comprehensively measure and analyze its process and products, and
- apply its process to new applications and provide continuous improvements and optimizations

To achieve level 5, each software organization must first apply the self-evaluation framework and take steps to achieve self-improvement. The panel recommends that: TTCP member nations encourage the use of the Software Process Maturity Framework by their internal software development and project management organizations, and by their software contractors and software subcontractors.

Recommendation 5. Establish metrics education and training

The TTCP member nations should establish Metrics Education and Training courses and manuals for Software Program/Project Managers. The courses should be part of an expanded curriculum in Software Engineering used by educational institutions (e.g., the U.S. Defense Systems Management College) to educate and train professional program managers and their supporting staffs. In addition, the SEI should assist in establishing these Software Engineering and Software Program Management curricula.

3.3.6 REFERENCES

1. W. Humphrey, Managing the Software Process. Reading Massachusetts: Addison-Wesley; 1989.

2. L. Osterweil, "Software Processes are Software Too," *Proceedings of the Ninth International Conference on Software Engineering*, 1987.

3. J. Valett and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Journal of Systems and Software*, Vol. 9, No. 2; February 1989.

4. T. C. Jones, Programming Productivity. New York, McGraw Hill; 1986.

5. R. Grady and D. Caswell, Software Metrics: Establishing a Company-Wide Program. Englewood Cliffs, NJ, Prentice-Hall; 1987.

6. COSMIC: NASA's Computer Software Management and Information Center, University of Georgia, 382 E. Broad St., Athens, GA 30602.

7. Valett, "The Software Management Environment," *Proceedings Thirteenth Annual NASA/GSFC Software Engineering Workshop*, 1988.

8. R. Taylor, F. Bell, L. Clarke, L. Osterweil, R. Selby, J. Wileden, A. Wolf, and M. Young, "Foundations for the Arcadia Environment Architecture," *Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments*, 1988.

9. C. Byrnes, "Formal Design Methods for Dynamic Ada Architectures," *Proceedings of the Eighth Annual National Conference on Ada Technology*, 1990.

10. V. Basili and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988.

11. R. Selby, G. James, K. Madsen, J. Mahoney, A. Porter, and D. Schmidt, "Classification Tree Analysis Using the Amadeus Measurement and Empirical Analysis System," *Proceedings of the Fourteenth Annual NASA/GSFC Software Engineering Workshop*, 1989.

3.4 Panel 3: Software Metrics Acquisition Issues

3.4.1 General Information

3.4.1.1 Panel Participants

NAME	EMPLOYER	COUNTRY
Robert Converse, Chair	Computer Sciences Corp.	USA
Judy Clapp	MITRE Corp	USA
Thomas Eller	Kaman Sciences Corp	USA
Kaye Grau	Software Productivity Solutions	USA
Jean-Claude Labbe	Defense Research Est, Valcartier	Canada
Michael Looney	Admiralty Research Establishment	UK
Thomas Triscari	US Air Force/Rome Laboratory	USA
Ray Walsh	Dynamics Research Corp	USA

3.4.2 Introduction and Overview

3.4.2.1 Scope

The Software Metrics Acquisition Issues Panel addressed issues concerning the use of metrics up to the time of contract award and the contractual mechanisms needed to ensure that metrics data is obtained and used during the period of contract performance.

3.4.2.2 Purpose

The Panel defined three areas to be addressed:

(1) The use of metrics to develop better criteria for the evaluation of proposals and the assessment of the abilities of the offerer leading to the selection of a winning contractor;

(2) The contractual mechanisms that are required to obtain and analyze the metrics to provide effective management/monitoring of the process and evaluation/quality assessment of the products including intermediate products;

(3) Recommend acquisition strategies and metrics that may be applicable to the acquisition process.

3.4.3 State-of-the-Practice

As a general statement of the state-of-the-practice for the use of metrics in current programs, the Panel felt that, at best, it was used informally, was considered to be a plus, but this use was not required, nor included on CDRL's.

3.4.3.1 Contract Strategies

A currently used contract strategy aimed at ensuring the selection of the contractor that provides the best value to the Government is a phased competition. With this strategy, multiple awards are made for a prototype or run-off competition and a single award is made to the winner of the run-off.

3.4.3.1.1 Contract Types

The following list contains the contract types that are typically used for software/system development efforts.

- CPFF-(Cost plus fixed fee),
- CPIF-(Cost plus incentive fee).
- CPAF-(Cost plus award fee)
- FFP-(Firm fixed price),
- FPIF-(Fixed price incentive fee),
- FPLOE(Fixed price level of effort)
- TM (Time and Materials)

Of these, the contract types for which incentives can be provided to the contractor are CPAF, CPIF, and FPIF. The panel felt that a FFP type could actually provide a disincentive because of the limitations on the amount of profit that a contractor can make.

3.4.3.1.2 Incentives

The amount of incentive to be paid is determined in various ways. For an award fee, the percentage of the maximum fee to be paid is normally determined subjectively; no formal metrics are used. On some contracts the award fee is distributed among members of the development team to ensure their participation on the project, thereby maintaining the quality of the product and the level of productivity. An incentive fee is usually applied to end-item deliverable projects where a specific minimum performance requirement is met on schedule and the amount of the fee is determined accordingly. This is not usually applied to software development contracts.

3.4.3.1.3 Current Methods for paying incentives

The methods that are currently used for paying incentives provide insufficient insight into the process or the quality of the product or they are applied too late in the process to effect corrections to the effort.

Acceptance Testing: Occurs at the end of the development effort and is a binary pass/fail decision.

Schedule: Based on contractually required milestones, rather than on an objective assessment of real progress.

Cost: Based on actual versus planned expenditures. rather than on the value of the real progress to date.

Traceability: The extent to which requirements are reflected in the design and implementation and the ability to show where an element of the implementation is required by the design or an element of the design satisfies a specific requirement. The traceability is usually shown via a matrix.

Subjective Assessment: Evaluation of a contractor's performance and the quality of the product.

3.4.3.1.4 Selection Strategy

A recent innovation for selecting a contract winner, but an approach that is gaining more widespread use, is to provide a software engineering exercise to the offerers and evaluate each offerer's performance on that exercise. This approach provides insight into each offerer's capabilities, but the evaluation techniques are still largely subjective and can benefit from the availability and use of effective measurement and analysis techniques.

3.4.3.1.5 Contractor/Subcontractor Relationships

The current approach for software engineering contracts that sub-contractors meet the same requirements as those contained in the prime contract is for the prime contractor to "flow down", or impose those same requirements on the sub-contractor. The prime contractor is then responsible to the Government for that sub-contractor's performance because, in general. the Government does not have detailed visibility into a sub-contractor's process.

3.4.3.2 Standards, Policies and Guidelines

Current use of standards, policies, and guidelines in the member nations ranges from no applicable standards which specifically address the issue of metrics apart from quality measures (e.g., the UK) to the use of a minimal set of metric standards (in the U.S., Australia, and Canada). Since Australia and Canada both make use of current U.S. standards and guidelines, the applicable U.S. standards and guidelines are addressed first.

Within the U.S. Department of Defense, the following standards with some applicability to metrics collection are commonly referenced in contracts:

- DoD-STD-2167A
- DoD-STD-2168
- MIL-STD-1521B
- Cost Accounting Standards

Earlier versions of these standards (e.g., DoD-STD-2167, MIL-STD-1679A) are still in use on older projects, but only the most recent versions of these standards will be discussed in this report.

DoD-STD-2167A and its associated Data Item Descriptions contain a few references to cost, schedule and risk tracking, quality and performance measurement, metrics and metrics collection. In the Software Development Plan (SDP), the manager must provide the initial schedule for the project. Most contracts require the update of the SDP at appropriate times throughout the contract so that any revisions to the schedule can be reflected in the SDP at that time. The manager is also required to describe the processes and methods that will be used to develop and track the development of the software in the SDP. The System/Segment Specification (SSS) and the Software Requirements Specification (SRS) require the analysts to specify system and software quality requirements. In the SRS, the software quality framework to be used for the project must be defined and the appropriate quality goals specified. In the Software Design Document (SDD), the engineer is asked to supply estimates of timing and sizing information. In the Software Product Specification (SPS). actual performance measurements of the delivered system are requested.

DoD-STD-2168 requires the delivery of a Software Quality Evaluation Plan which describes how the contractor plans to evaluate the functionality, performance, and qualities of the software under development.

DoD-STD-2167A references several other standards, including MIL-STD-1521B, which defines how reviews are to be held and the content of the review materials. MIL-STD-1521B requires the review of performance estimations at several of the reviews.

In addition to the technical standards, every government contract requires some form of cost accounting on a regular basis. generally monthly. This information provides the basis for contractor billing. Contractor's cost accounting systems are certified by the government through DCAS and the accuracy of the cost information must be personally certified by contractor personnel.

Other than the cost accounting DIDs, there are no known approved DIDs for requiring the delivery of quality or performance metrics on a regular basis.

Within the U.S. DoD, most software development contracts require the use of DoD-STD-2167A. Some require DoD-STD-2168. All require some form of cost accounting. In Canada, DoD-STD-2167A is also being used as a standard on military contracts. Australia is also using DoD-STD-2167A and its associated standards on military contracts. In addition, an Australian standard, AS 3563 Software Quality Assurance, provides a high level guide to setting up a Software Quality Assurance organization within a company. It does not specifically address metrics or metrics collection.

In addition to the applicable military standards, some U.S. guidelines more specifically address metrics and metrics collection:

- AFSC/AFLCP 800-5, Software Independent Verification and Validation
- AFSCP 800-14, Software Quality Indicators
- AFSCP 800-43, Software Management Indicators
- AFSCP 800-45, Software Risk Abatement
- AFSC/AFLCP 800-51, Contractor Capability Assessment (not yet released)

AFSCP 800-14, Software Quality Indicators, recommends a set of quality indicators (i.e., completeness, design structure, defect density. fault density, test coverage, test sufficiency. and documentation), how to collect the data and how to interpret and use the quality indicators. The recommended process is basically composed of four steps:

(1) Establish desired goals or thresholds

(2) Capture and plot the data

(3) Analyze the trends to determine the degree of goal achievement and achievability of the goal

(4) Implement corrective actions and reassessments of goals as the system matures.

AFSCP 800-43, Software Management Indicators, defines a set of management indicators for computer resource utilization, software development manpower, requirements definition and stability, software progress--development and test, cost/schedule deviations, and software development tools. It also describes how to collect the data and how to use and interpret the indicators.

AFSCP 800-45, Acquisition Management Software Risk Abatement, provides a risk abatement framework accommodating both qualitative and quantitative risk assessment. For each software risk area (namely performance, support, cost, and schedule), a brief synopsis of the types of information needed, the analysis required, and the risk reduction behavior expected is provided. Rules of thumb useful for identifying, analyzing, and handling risk are also identified.

AFSC/AFLCP 800-51, Contractor Capability Assessment, has not yet been officially released but is available from the Software Engineering Institute (SEI) as CMU/SEI-87-TR-23. This guideline provides guidelines and procedures for assessing the ability of potential DoD contractors to develop software in accordance with modern software engineering methods. It includes specific questions and a method for evaluating the results. Some of the metrics referenced in the maturity level questions include software size, cost:, actual and planned staffing, design, code and test error statistics, unit test completion, unit integration, memory utilization estimates and actuals, throughput utilization estimates and actuals, I/O channel utilization; software trouble report and action item tracking, design and code review coverage, test coverage, process metrics and software productivity.

In addition to the applicable standards and guidelines, the RL Software Quality Framework (RADC-TR-85-37, Specification of Software Quality Attributes) has been required as a metrics collection framework on several contracts within the U.S. Air Force and Army. A joint effort between STARS (Software Technology for Adaptable, Reliable Systems) and RL produced a set of Software Data Collection Forms (available from DACS, #7102-7109) which serve as the Software Quality Framework's metric element database.

3.4.3.3 Experiences

3.4.3.3.1 Cost

Most contracts do not explicitly price metrics collection.

In the case of the ESD/MITRE Software Management Indicators, there are model Statement of Work paragraphs and Data Item Descriptions available for collection of the indicators, but these have not yet been regularly used, and no data are available on the cost of data collection and reporting. An estimate of 7.5% - 8% of development cost was cited for metrics collection in the NASA Software Engineering Laboratory experiments.

A brief survey of the programs using ESD/MITRE Software Management Indicators showed no significant difference in the quality of the data and the extent of its use between contractors for whom there was explicit tasking for metrics and other contractors.

Metrics cost has been contained by tailoring the metrics to the contractor's development process and ability to provide the data.

3.4.3.3.2 Benefits

Current metrics, except for cost accounting data, are most effective when used as indicators of potential problems and not as objective measures of performance or progress.

In acquisition, the most rigorous metrics have been those collected for monitoring cost. These are managed separately, and the contractor's accounting system and methods of reporting are certified. For other software metrics, the most effective use of metrics has been to observe trends, to compare planned against actual status and performance, and to use the information as an indicator that some area needs further investigation. At this point, experts should be called in to apply their judgement as to whether there is really a problem, and what is cause and cure.

Metrics have more impact when the contractor as well as the government uses them.

When the contractor merely reports metrics because the government requires them, then neither the government nor the contractor is likely to benefit from them.

Most of the focus of current metrics in use is on benefitting an individual program, rather than collecting information to benefit future programs.

Metrics data provided to the government are primarily for the purpose of monitoring a specific program's cost, schedule, and performance. There has not been a significant effort to preserve the data in order to provide a basis for predicting these metrics on subsequent programs or improving software acquisition.

The greater the government involvement in metrics collection and analysis, the greater the benefit to the government.

In cases where the government has been involved in the data collection process, there has been a benefit in greater understanding of the data and in greater attention to the metrics. When the government has collected basic data and performed analyses tailored to its use of the data over time, the data have been more useful. The notable example of this is NUSC, where there are government resources dedicated to analyzing data and producing the metrics in direct response to management issues.

3.4.3.3.3 Problems

Most metrics provide information which is not timely enough for a decision maker to affect the course of the program.

The most prevalent metrics in use for measuring progress and product quality are the error reports during testing. Unfortunately, in a traditional waterfall model of development, these are the last stages of the development process. Other metrics merely show that a problem has already occurred. rather than providing early warning, e.g., program complexity can only be demonstrated when the software has been written.

There is often a lag in the availability of metrics data to the decision maker that prevents timely decisions. By the time metrics data reach the decision maker in the government, it may no longer be timely. Thus, remedial action may have been precluded, or the consequences of the delay have been to aggravate the problem.

Even when metrics are collected, they are not effective because they are not believed or understood by managers.

Both contractor and government managers need to be convinced of the utility of metrics to them. When a problem erupts on a program, it has been possible to reexamine the metrics and find that the evidence of a problem was there.

Most metrics are not sufficiently validated to be used for anything more than indicators.

Metrics which are used to estimate or predict the outcome of a development and the quality of the process have not, for the most part, been sufficiently validated to support significant actions on the part of the program's management. The typical normative data over a statistical sample lack the rigor in definition of the conditions under which they were collected or apply to a context which is sufficiently different from the current program to make comparisons invalid. This seems to be true of productivity data and error densities. which are often cited as a basis for assessing a specific program.

When metrics are collected by IV&V contractors, they are not always shared with the prime contractor.

IV&V contractors can be used to collect metrics data The benefits realized from metrics are strongly related to who sees the metrics. One of the principles followed by

Japanese companies in the collection and use of productivity and quality metrics is that the data should be collected and presented to those who are in a position to act on the results. If metrics data from the IV&V contractor is not shared with the prime, then the prime cannot adjust his activities in accordance with the feedback.

3.4.3.3.4 Lessons Learned

Metrics need to be integrated into the management process if they are to be used to predict or avoid problems.

At the start of an acquisition, program management must determine what purpose metrics can serve. A risk assessment can be helpful in identifying priority measurement targets. Both the acquisition process and the metrics should be specified to provide the kind of information needed to manage risk at the earliest possible time for making it available. Selection of metrics should be a joint decision of the developer and the government. Resources should be allocated by both organizations to the collection, analysis, presentation, and use of the metrics that are selected.

Metrics alone may provide an indicator of a problem if they are used, but they do not provide a solution nor do they indicate the problem's cause.

More than the mere collection of data is needed to make effective use of metrics. Resources must be allocated to understanding the data, and its causes. Looking for solutions requires experience and judgement.

There can be problems in imposing metrics on subcontractors that conform to the metrics provided by the prime contractor.

Contractually, a subcontractor may not be required to report metrics to the government, and the prime may not be required to levy such a requirement on the subcontractors. If the contractor is small, he may be unequipped or unable to afford to collect metrics. The prime may not have a meaningful way to integrate his metrics data with that of the subcontractor(s), depending on how the work has been allocated.

Often different requirements are levied on contractors for cost reporting and for productivity reporting, which causes two sets of books to be kept.

Cost reporting for labor may have different requirements than the reporting of metrics for software productivity. To meet the requirements, two different accounting systems are needed for labor, which adds cost to the metrics reporting.

3.4.4 State-of-the-art

3.4.4.1 Emerging Contract Strategies

The panel agreed that in some cases the procurement agencies are tailoring the metrics. An appropriate set for the size and objectives of each software acquisition is sometimes included in the contract. This approach can result in a cost saving by not requiring unnecessary metrics to be collected and analyzed. The panel recommended that a single reference be created for all metrics and that it serve as the foundation from which the metrics are tailored.

The panel noted that metrics should not be limited to the DoD-STD-2167A waterfall approach to software development. The state-of-the-art for software development now includes rapid prototyping and spiral development models. In some cases, metrics can be used across the emerging software development approaches. The panel agreed that metrics that cannot be used across emerging approaches should be so identified.

Current approaches to obtaining metrics data is to require the information be provided under the Data Item Description (DID) for Conference Meeting Minutes and by developing Unique Data Items. The UDI approach was determined to be very difficult. This was because the contract authority resisted use of UDI's as a general practise. In the future a DID should be developed for the reporting of metrics data.

There was one case brought to the panels attention where the procurement agency had electronic access to the contractor's metrics formation. While this appeared on the surface to be a good idea there was some concern that such access would encourage micro-management. However the benefits of having the metrics data available in real-time may well be worth the risk of micro-management. If both the procurement agency and contractor pays careful attention to using the metrics consistent with their individual objectives it would be possible to achieve each organizations goals.

Several cases were described where the software developers were required to use a Computer Aided Software Engineering (CASE). The compliance with this requirement resulted in a binary metric (yes/no). The panel believed that significant improvements could be achieved if metrics were developed to quantify characteristics of the software design as it is portrayed in the CASE tools.

Through the Software Engineering Institute (SEI) contractor assessment process, there is a method of determining the metrics collection, analysis and reporting capability. The policy of assessing contractor's software development capability is not being applied across all DoD software procurement. Such a policy should be established.

A few contract strategies allow for the contractor(s) to proceed to refine the requirements and in some cases provide preliminary design solutions with operational prototypes prior to entering into a fixed cost contract. The metrics to be collected during the system maturation period were not known but it appeared that a class could either be tailored or developed to help measure the progress of the contractor.

The SEI continues to maintain an initiative in the software metrics area and their results should be provided to The Technical Cooperation Program (TTCP) software committee

3.4.4.1.1 Contract Types

The state-of-the-art for use of metrics for determining contract type is the same as the state-of-the-practice information provided elsewhere in this workshop report.

3.4.4.1.2 Incentives

The state-of-the-art for metrics use in determining incentives is the same as the state-of-the-practice, information provided elsewhere in this workshop report.

3.4.4.1.3 Metrics for "paying" incentives

The panel members knew of a few contracts that provided the software developer with financial incentives based on measurements. The incentives were based on specific objectives that the procurement agency wanted to achieve. The measurement could be objectively made by the software developer and validated by the procurement agency. The specific metrics being used to award incentives are as follows:

Reuse - The number of Computer Software Components (CSCs) that were reused in the developed software. The panel agreed that this may be a legitimate goal for the procurement activity and the reuse of CSCs may result in a cost saving. On the downside, the software developer may be encouraged to reuse CSCs that are not as appropriate as redesigned CSCs would have been for the system. The risk of this occurring could be reduced by reviewing the design documentation for the reuse CSC candidates and determining based on the design data the appropriateness of their reuse in the software.

Software Defects - The number of Problem Trouble Reports (PTRs) discovered during the course of developing and delivering the software. The panel believed that rewarding the software developer for having fewer PTRs should encourage the software developer to spend more time on the design and other activities that would result in better code. The measurement of the PTRs should be clearly stated as well as the method for validation. The panel concluded that this was a reasonable approach and with careful attention by both the procurement agency and the contractor both of their objectives could be achieved.

Warranties - The terms of this incentive approach were vague. It appeared that in one case a contractor was awarded a premium if the delivered software operated over a period of time without problems. The panel believed that this approach would be difficult to administer. Warranties should follow the traditional approach - the developer is responsible for fixing problems for a certain period of time. Superior software development contractors will be rewarded by not being required to use the reserved funds set aside for warranty work.

3.4.4.1.4 Selection Strategy

The state-of-the-art for use of metrics as a selection strategy is the same as the state-of-the-practice information provided elsewhere in this workshop report.

3.4.4.1.5 Contractor/Sub Contractor Relationship

The state-of-the-art for use of metrics in the contractor/subcontractor relationships is the same as the state-of-the-practice, information provided elsewhere in this workshop report.

3.4.4.2 Standards, Policies, Guidelines

The state-of-the-art standards, policies, and guidelines for metrics in the acquisition phase are the same as those found elsewhere in this report with the addition of the guideline bring prepared by the RTCA Special Committee 152 (Software Considerations in Airborne Systems and Equipment Certification), Technical Report RTCA/D0-178A, Radio Technical Commission for Aeronautics, March 1985. This guideline is currently under consideration by RTCA special committee 167 which is tasked with revising D0-178A as necessary. The role and use of software metrics is being addressed by this special committee and the draft is due in fall 1990.

3.4.4.2.1 Issues Regarding Application of Standards/Guidelines

IEEE has developed several standards and guidelines that overlap with those used by the DoD. However, the material appears to be in some cases supplemental to the DoD direction. The content of the IEEE information should be reviewed and appropriate information incorporated into the DoD metrics approach.

3.4.4.3 Known Future Directions

The panel noted that there are efforts underway to establish metrics data bases that will support collecting, analyzing, and reporting metrics throughout the software life-cycle. In certain situations, the data can be collected from the output of tools but this appeared to only apply to measurements made on the code. To support the program manager requirements the metrics database must be capable of ad-hoc queries for data relevant to software program areas of concern.

Rules have been developed by the SEI to generalize the software engineering information obtained from assessing contractors to protect the contractors identify yet allow the use of the data in assessing the state-of-the-practice for the software industry. This approach needs to be extended to the metrics data to allow analysis over a broad range of software development without releasing sensitive metrics data on either specific contractors or programs.

3.4.5 GENERAL ISSUES:

The purpose of employing software quality metrics is to provide appropriate levels of management with the information and insight required to make decisions and take action in sufficient time to prevent or minimize undesirable impacts on cost, schedule or performance goals. Several issues of a general nature were identified and discussed that influence the implementation of software quality metrics within acquisition programs. The issues provided a backdrop to understanding the state-of-the-practice and the barriers needed to be overcome in order to advance the state-of-the-art.

The general issues are:

- Requirement Change
- Validation of Metrics
- Cost for Metric Data Collection
- Time Lag for Reporting Metric Data
- Protection of Sensitive Data
- Personnel Career Development
- Uses of Acquisition Metrics

3.4.5.1 Requirement Change

Software-intensive development efforts tend to experience requirement changes at a rate that exceeds the system development cycle. These changes complicate the reporting, analysis, and interpretation of software metric data, making more difficult the identification of relationships and trends. Further, changing requirements confounds the effect of management action resulting from the use of the metric information.

3.4.5.2 Validation of Metrics

Face validity of software metrics has not been sufficient to motivate their use. Empirically based validation of software metrics must be performed to provide a foundation to build support for their use.

3.4.5.3 Cost for Metric Data Collection

Key to the use by management of software quality metrics is the belief in their value added. The cost added of reporting,analysis and interpretation of software metric data needs to be established as well as the benefit added.

3.4.5.4 Time Lag for Reporting Metric Data

A potential problem in achieving an effective software quality metric system is the time delay in reporting information from the programmers through various levels of management. Although automation of data collection may in part resolve this problem, another may be introduced by the capability of such a system to allow micro-management to occur (i.e., providing too detailed information to the program office).

3.4.5.5 Protection of Sensitive Data

This issue relates to the concern of releasing metric data that may result in any of the following situations:

- loss of competitive advantage
- misuse of data by higher management

3.4.5.6 Personnel Career Development

The career development and progression of personnel engaged in software quality was identified as an issue of concern. Questions raised were:

- Is it necessary to have experience as a computer programmer in order to perform as a software metric analyst?
- What skills, education, experience should a software metric analyst have?
- What is a logical career progression for such personnel?

3.4.5.7 Uses Of Acquisition Metrics

Acquisition software metrics may provide assistance in developing acquisition strategies in the following ways:

- Determine appropriateness of type of contract, incentive/award fees, etc. over the development life cycle.
- Assess the completeness and adequacy of the RFP
- Assessment of the quality of the specification
- Personnel qualifications needed

3.4.6 Recommendations

3.4.6.1 General:

We believe that the mere addition of metrics to the current state-of-the-practice of software acquisitions will not significantly improve the quality of the software produced. Rather, a fundamental change to the acquisition process is required to decouple the build-the-system (hardware and software) phase of the acquisition from the shackles of the contractor's cost estimate made during the proposal activity. This decoupling is required to give the contractor/government team the freedom to implement the changes in requirements and design that naturally will be revealed during the design phase. This is especially true if the team is to take advantage of rapid prototyping and other risk reduction processes, and of the insights that can be gained from metrics.

This fundamental change along with supporting recommendations are explained below.

3.4.6.2 Acquisition Approaches:

1. Base milestone completion on quality criteria rather than on schedule and cost.
2. Use objective incentive type contracts (FPIF for certain well defined phases, or CPIF in general) rather than fixed fee or subjective award fee contracts. This will make the product metrics integral to the acquisition by giving the contractor an incentive to improve the quality of the software.
3. Use the Process Maturity Levels from the SEI Contractor Assessment Methodology both as a part of the Source Selection Process as well as periodically during the life of a program as a means of tracking the maturity growth of the contractor team.
4. Split the acquisition into two distinct phases: a CPIF type contract for the requirements and design phase, and a CPIF or FPIF type contract for the build phase, where the cost of the second phase is not estimated until the results of the first phase are known.

Present acquisition is structured around a single contract placed to cover the stages from requirements analysis to implemented code. The process is estimated for cost and schedule using source lines of code (SLOC) as the measure on which the development process is based. The SLOC metric is not reliable as a means of estimating the development schedule, yet it appears to be the dominant metric used in the present cost models and tools.

If lines of code is to be considered as the metric for estimating the coding and testing phase, which may be considered an acceptable point at which to use it, then it is necessary to have a more accurate set of information on which to base the SLOC value. To provide this, more detailed values should be obtained from the design and requirement stage which can be used to identify the size of the final code.

To enable this to be achieved two approaches are recommended. First, there should be more use of the design metrics which have been proposed in various areas. These, although not validated at present, should be used to carry out more evaluation of design in order to provide the necessary metrics which can be used for the following stages of the development. This process should also be applied to the requirements stage, but as of yet there appears to be no metrics associated with this part of the development. Research is needed in this area to identify which metrics can be collected and which ones are important.

The second point to be made is that the acquisition process does not distinguish between the front-end process and the back-end development. At present, most contracts are awarded for the entire development effort, with no distinction between what should be an interactive process at the point when the requirements are being clarified and the coding. If the contract is split and there are two stages to the procurement, then there are considerable benefits to be gained.

The first stage of the acquisition should look at validating the requirements, using prototyping to achieve this with the end user being involved in the process of assessing the acceptability of what had been produced. There would be a need to control/manage the use of the prototype in whatever form it is used. This should also include identification of the intention either to discard or carry forward all or part of the prototype. The objective of this contract would be to demonstrate to the end user that his functional requirements were correct and that his nonfunctional requirements are achievable. This should cover down to and include top level design, which may have significant impact on the achievability of the requirements laid down.

The first contract stage should be more flexible in that it will require interaction with the end user to resolve the ambiguities and issues arising from analysis of the initial requirements. This may require cost plus contracts to allow for coverage of the more open ended luster associated with refining and clarifying the requirements to the necessary level at which they can be used to 'build' the system. This process may be iterated several times with 'incremental builds' taking place to provide initial deliverables/systems for in-service use.

The second stage of the process should be a contract to build the system, covering the detailed design and coding based upon validated output from the first contract stage. This stage should be mechanistic, as far as possible, using application code generation and any 4th generation type languages where appropriate.

The structure for the local development process would be based on a spiral structure from which the builds would spin off as various releases, with the two stages being covered by the separate contract types.

Once completed, the output from the second stage becomes a deliverable and any modification/enhancement should be input to a repetition of stage one, where impact analysis and change control are carried out.

Over the years, the level of abstraction has moved upward away from machine code and assembly language systems to ITOC's and code generation, but the contract acquisition process has changed minimally. Now that there is an ability to take the development to an even higher level of abstraction by the adoption of a prototyping approach, it is essential that the acquisition process should be modified to accept the changes. The emphasis on the front end is essential and has been accepted, yet the procurement process appears to ignore this. The proposed changes would move a long way towards accepting this change of emphasis while increasing the understanding and improving the correctness of system requirements prior to commitment to build. It will also remove the reliance on the preproposal SLOC metrics for the estimation of the time and cost to develop the whole system since the results of the requirements and design stage can be clearly seen and the second stage contract can carefully be placed using a fixed price approach.

3.4.6.3 Validation Approaches for Future and State-of-the-Art Metrics:

It is essential that the set of process and product metrics that will be required as a part of government software acquisitions be credible. Managers and practitioners in both government and contractor communities must believe that the use of metrics will improve the quality of the software acquired. Two basic activities will provide data to help establish this credibility.

1. Establish metric data collection standards, contract language, and a central repository for storage, analysis. and distribution of summary information. The latter already exists through Rome Laboratory's Data and Analysis Center for Software (DACS). The standards and contract language to produce a sufficient quantity of high quality data remains to be done and will require resolve, funding, and continued management attention.

2. Produce Case Studies and Conduct Experiments. Case studies when coupled with controlled experiments on sufficiently large acquisitions will provide needed calibration to aid in interpreting the generalizations which will stem from the vast amounts of data that will accumulate through the DACS.

3.4.6.4 Technology Programs:

Several new programs are needed to facilitate the use of metrics to improve the quality of software acquired by the government.

1. Develop an integrated set of acquisition package items to insure that the structure and conduct of the acquisition can take advantage of the state-of-the-art risk reduction approaches including the use of metrics.

   It is fundamental that the various parts of the contract must play together for the process to work. It is the state-of-the-practice that acquisition packages are frequently pulled together from various previous contracts and often hinder rather than help the production of high quality software.

   Thus, it is essential that mutually consistent words for the Contract Schedule, Terms and Conditions, Representations and Certifications. Proposal Instructions and Evaluation Criteria, Specifications. Statement of Work (SOW), Applicable Documents List, Work Breakdown Structure (WBS), Contract Data Requirements List (CDRL), and Data Item Descriptions (DIDs) be provided to all agencies acquiring software and to the various government agencies (DCAS, AFCMD, AFPRO, etc) auditing and monitoring contracts.

   These contract words must mandate the collection and reporting of the required software metrics in a manner that is consistent with already mandated cost/schedule reporting definitions and requirements.

   The guidance associated with this model contract language must mandate that the standards, DIDs, etc. be tailored to fit the needs of the particular acquisition.

2. Develop guidelines/handbooks for collection and analysis of metrics. The guidelines that exist today are contained in numerous publications that are redundant and sometimes inconsistent. Therefore, AFLC-AFSC Pamphlets in the 800 Series (and equivalent publications of other Services, Agencies, and Governments) need to be

consolidated into a concise, unified treatment of product metrics, management indicators, and risk reduction techniques that are consistent with the acquisition words discussed in item 1 above.

3. Develop clear guidelines for tailoring. This will help mitigate the fear and ignorance that leads to the *I need all of the paper* syndrome.

4. Develop criteria and metrics that can be used to measure the quality of the Request for Proposal (RFP)/Acquisition Package. The entire life of most contracts is determined to a great extent by the appropriateness, consistency, and quality of the Acquisition Package. This should include an Acquisition Office Checklist similar to the SEI Contractor Checklist.

5. Develop metrics and analysis methods that can be used during the requirements and design phases of an acquisition to measure both the quality of the requirements and the design and their rates of change. Most of the product metrics proposed today are useful only after coding and testing has begun.

3.4.6.5 Other

1. Train Commanders, SPO, DCAS, AFPRO, GAO, Program Executive Officers, and Contractor management personnel in the proper use and interpretation of both process and product metrics and in the proper application of the modified acquisition approach and consistent contract language described above.

2. Ensure that mandated software metrics definitions, collection, and dissemination rules are consistent with the rules governing cost/schedule reporting.

   This is essential to avoid both the leakage of proprietary data and source selection sensitive information as well as the inevitable waste of resources that attends the inevitable cycle of suspicion, allegation, animosity, and sometimes litigation that stem from apparent inconsistencies in workload/productivity data collected thorough different routes.

3. Develop and use criteria and metrics to clarify the use of Commercial Off-the-Shelf (COTS) products as a part of software acquisition.

   COTS is usually thought to imply that it has all the benefits of good old capitalism in the market place--that it has been rung out by a large base of users, thus insuring its quality, and that the cost of its development has been shared by all those users. Thus the government avoids cost, schedule and quality uncertainties. The fact is that the definition of COTS is so broad that these benefits are not assured.

4. Devise a metric to measure the change in requirements that are affecting the acquisition.

3.5 Panel 4: System Metrics

3.5.1 General Information

3.5.1.1 Panel Participants

NAME	EMPLOYER	COUNTRY
Joseph P Cavano, Chair	US Air Force/Rome Laboratory	USA
Walter R. Beam		USA
Samuel DiNitto	US Air Force/Rome Laboratory	USA
Eugene Fiorentino	US Air Force/Rome Laboratory	USA
Paul Szulewski	Draper Laboratory	USA

3.5.2 Introduction and Overview

The intent of the System Metrics Panel was to capture the big picture viewpoint of a project by primarily addressing the system level and then identifying the interfaces to hardware and software. We treated the focus of this panel in terms of answering the following questions: Does the system do what it is supposed to do? What are the critical factors that the system must exhibit in order to be considered a success? In order to begin to answer such questions and define what might be critical system factors, we first had to establish the scope of our panel.

We used the project level to represent our viewpoint of a system (the Process Metrics panel referred to this as the big P level). A system combines things or parts into a unified whole to provide a solution to a particular problem; system architecture is the arrangement of system components based on decisions to allocate certain functions to hardware components and software modules. We recognized that a system must provide some functional capability (solve a problem), but we did not attempt to express metrics that could only be project specific; i.e., relate only to the functions for a given application. Rather we focused our attention on quality issues related to any system. Our intent was to describe metrics which characterize a system as opposed to metrics which characterized a system's constituent parts. We sought metrics that could be related to meaningful goals for a user, customer or acquisition manager, that would be clearly understood, objectively measurable, and appropriate for both hardware and software.

We chose mission requirements as the inputs for our panel's domain. We used them as the starting point for system definition, concentrating on a customer or user viewpoint (these terms are interchangeable in this text). If we found ourselves forced to *distinguish between hardware and software when considering mission requirements*, we probably were not thinking at the system level. We proposed six system factors that were most appropriate or most critical to the success of a system: system reliability, maintainability, digital performance, extendibility, safety, and usability. We attempted to characterize these system factors beginning with mission requirements and extending across the development life cycle, always with the idea of showing whether or not customer requirements were being met.

We did not address functional requirements directly because they are application dependent. However, we chose the context for our system factors in terms of meeting functional requirements. For example, system reliability (regardless of how it is expressed) is ultimately concerned with meeting functional requirements defined in terms of the probability of successfully completing the mission.

Given the mission requirements, we wanted to show quantitatively how those requirements were being met. At the same time, we were concerned with being able to use system factors throughout the development process in order to help implement continual quality improvement. Recognizing the many tradeoffs to consider and the interrelation of many processes, we sought global metrics that represented a system perspective for developers, regardless of whether hardware components or software subsystems were of concern. In order to achieve such a global description required compromises, because hardware and software cannot always be treated in a similar way (i.e., in common units). However, our compromises were motivated by the desire to be able to show that mission requirements were being met, and this could best be done from a system perspective. Typically, users are not concerned with how reliable individual software or hardware components might be—they want to be certain that the system is available and can accomplish the mission without failure of any kind. At the same time, requiring that hardware and software be described in a similar way in order to show conformance to mission requirements does not preclude treating them separately at some lower level within their own specialized domains.

When faced with a choice for providing depth or breadth, we strove for breadth, leaving for the other panels to consider how to implement the depth. We wanted to describe system factors without having to distinguish between hardware or software, and if such a distinction had to be made, we sought only to identify it for other panels to address.

We considered, but excluded, other potential system factors. For example, we considered availability but decided not to include it because it could be computed from two other already identified system metrics, mean time between failures (MTBF) and mean time to repair (MTTR). Under conditions where continuous system operation is required and no preventive maintenance is carried out, availability is simply described the ratio of uptime to uptime plus downtime which, in turn, can be defined as

$$\frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Similarly, candidate system factors which we believed would not be of direct concern to a user, were also excluded from further consideration. For example, a potential factor such as fault tolerance was treated as a technique for achieving required reliability, not as a direct system requirement. It seemed to us that a customer should not be interested in specifying or determining the fault tolerance in a system, but rather should be interested in objective measurements demonstrating conformance to reliability or availability requirements. (Both hardware development and software development can and should consider fault tolerance and relate it to achieving the desired reliability or availability.)

Above all, we wanted system metrics that are useful for monitoring, controlling, and modifying the system development process. If a metric would not help assessment of product quality, or help managers make good choices for accepting a product, or help developers to set recognizable and measurable goals to guide the development effort in meeting mission requirements, then the metric was not considered. Again, compromises were necessary and conflicts arose frequently. Our guiding principle was to choose measures that were easy to understand, easy to quantify, and easy to use .

Section 3.5.3 of the System Metrics Panel Report discusses state of practice, Section 3.5.4 defines system metrics at a high level, and Section 3.5.5 introduces a conceptual model to explain the use of system metrics within the context of a development life cycle. Recommendations are inserted at various points so that the reader can see the context in which they are proposed and then summarized in Section 3.5.6.

3.5.3 State-of-the-Practice

There is little difference between state-of-the-practice and state-of-the-art for the system metrics we have proposed. For projects adhering to DoD-STD-2167A, system and quality requirements should be identified in the System/Segment Specification and in the Software Requirement Specification. We believe it would be advantageous and preferable to identify system and quality requirements as early as possible in the acquisition cycle, i.e., in the Request For Proposal.

Recommendation 1. System quality requirements for reliability, maintainability, digital performance, extendibility, safety, and usability should be identified as part of the Request for Proposal (RFP) and should reflect the customer's inputs and concerns.

This section briefly describes how complete systems might be characterized today by sets of metrics for each of the system factors proposed.

Typical practice in specifying system reliability requirements is to define one or more mission operating states and their associated mission times and then allocate the metric mean time between failures (MTBF) to the average time the system must remain in operation without failure. Certain assumptions are also made regarding the independence of failure and their rate of occurrence. An appropriate reliability model is used to translate MTBF and mission time into a probability of successful mission performance. The system is then decomposed into its various functional components and the system MTBF requirement is allocated or budgeted to each component in accordance with its criticality to mission performance. Combinational relationships are well understood for hardware components but not for software units. Although software reliability is being treated more seriously now, past projects have either ignored software's contribution to reliability or treated it as perfectly reliable.

3.5.3.1 Maintainability

Maintainability is a factor used to express the ease with which maintenance can be conducted. Measures for maintainability include Mean Time To Repair (MTTR), probability of restoring service in a specified time period, and mean time for scheduled maintenance. However, there may be additional requirements necessary to specify maintainability completely. Some examples include specifying that the identity of a failed circuit board must be automatically indicated or that any replaceable part must be accessible within a certain time by an average maintainer with certain specified equipment. The satisfaction of such requirements normally require in-plant demonstrations prior to delivery to the customer. Closely related is the concept of availability—the ability of a system to perform its intended function when required for use. Availability is a function of both the reliability and maintainability of the system. To achieve high levels of availability, trade-offs among reliability, maintainability and cost must be made.

3.5.3.2 Performance

System performance may be described in terms of

- precision achieved
- rate of operation (e.g. cycles per minute/hour/day, etc.)
- probability of success in performing an operation under conditions of uncertainty
- rate of expenditure of fuel, oil, or other resources
- attainable force, speed or acceleration, or other physical (motion) parameters
- delay time between a stimulus and the resultant response

All of these are amenable to physical measurement, though they may be expressed in statistical terms if there is uncertainly.

3.5.3.3 Extendibility

Extendibility may be addressed by r qu:ring certain system capacities to be greater than required for the initial system. This may include electrical power, computer processor power or memory size, size of lubricant reservoirs, etc. Accordingly, the excess capacity (ratio of actual to minimum required capacity) is a common extendibility metric.

In software controlled systems, as in many other systems, extendibility requires not only excess capacity, but also a structure which can readily accept extension. Personal computers, for example, contain "slots" for printed circuit boards used to expand system function. Certain software controlled system such as calculators and laser printers may provide "slots" into which modular software may be placed in the form of packaged need-only memory in a cartridge.

3.5.3.4 Safety

Safety metrics appear in wide variety. There are three likely definitions of safety related to the avoidance of injury: 1) to system users, 2) to bystanders, or those in the path of the system, and 3) to the system itself. Typical quantifications of safety have been time–related; e.g., number of lost–time accidents per hours of exposure. Some systems may require that external parts never exceed some limited temperature; others may require installation of safety equipment within a given distance of certain system components. Safety concerns closely related to software may, for example, require that no accidental set of inputs be capable of triggering a particular safety fault.

3.5.3.5 Usability

In system practice, usability is most often addressed via comparative experimental evaluations, using representative users. In con:pa·'ng an older system design with an improved one, users may carry out (or simulate) corresponding complex cognitive and manual task sequences on both system. The speed, correctness, or fatigue over repeated operations of a user may be specified and compared quantitatively. The amount of training needed to teach a user and the type of skill level required to operate the system must also be considered. Other usability metrics may reflect research in human factors; e.g., requiring that characters on a display screen subtend a certain minimum angle at the viewer position, or requiring that the force to actuate a manual push-button be no greater than some given value.

3.5.3.6 Cost

There are two other important system factors, acquisition cost and operating cost. These factors are usually derived from other system characteristics and already receive a great deal of attention. Though there are many issues, such as whether maintenance equipment costs should be included in system acquisition costs, metrics are relatively straightforward for cost (dollars and cents). Cost accounting systems are in widespread use and are required in all contracts.

3.5.3.7 Schedule

Likewise, development schedules are subject to differences in interpretation because of overlapping of activities or non-inclusio: of some activities, but the metric is basically a time measure (dates and months). Initial software schedules for a project are provided in the Software Development Plan (under DoD-STD-2167A). The panel concentrated its efforts on the quality related system factors and did not consider cost and schedule in any depth.

3.5.4 Metrics for Software Quality

This section defines the set of system level quality factors introduced previously; the goal is to define the factors so that they can be characterized by quantitative, measurable attributes of the system, as a result of either process or product. Many attributes of system quality, as perceived by the user, are not reflected in the statement of the system requirements in a manner that facilitates measuring the absence, presence, or degree of conformance to these requirements at various points in the system life cycle. Although these measures are not perfect (and the list of factors may be incomplete, we hope that a set of system measures will help provide better products for the customer. Relationships of these measures for system, hardware and software are illustrated in Table 3.

Recommendation 2. Further research should be conducted to determine a complete set of system factors and appropriate measures for them.

In discussing system factcrs, we try to illustrate the kind of language required in the specification of systems to enable system-level metric data to be identified and tracked. They are intended to represent a global, system perspective—meaningful to the customer as well as the developer or manager who must meet the customer's requirements. Acquisition, product and process metrics (subjects of other panel reports) should relate to these metrics but are free to extend or augment them in their respective domains.

System Factors	System	Hardware	Software
Reliability	Human Factors	Ergonomics	Display Aesthetics
Maintainability	Criticality	Behavioral Driven	States causing Accidents
Digital Performance	Man-Hours	Excess Capacity	Enhancements
Extendibility	User Oriented Time	MIPS	Algorithm Efficiency
Safety	Man-Hours	MTTR	MTTR
Usability	Mission Time	Operating Time	Execution Time

Table 3. Factor Orientation to System, Hardware and Software

3.5.4.1 Reliability

Reliability addresses the probability that the system will successfully complete its intended mission, in the specified time and under specified conditions. For a system with software, the probability is a function of the inputs to,and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered (see IEEE P982.1). There are several figures of merit for reliability: Mean Time Between Failures, failure rate (number of failures per unit time), and probability of success.

System reliability parameters such as Mean Time Between Failures (MTBF) and software reliability models attempt to describe, in a global sense, the system's capability to successfully complete a specific mission. In an ideal world, MTBF would be equally suitable for hardware and software. If both hardware and software reliability were only concerned with initial verification and validation, a single metric might be easy to identify/define. However, hardware must wrestle with the additional problem of indicating ongoing failure–free operation; software is infinitely durable. Acceptable definitions of MTBF for software must be developed. Failures do arise in software operation when the right "input" uncovers a resident design or implementation error, so describing the number of software failures over execution time by means of MTBF may be a feasible approach even though time is not directly driving the failures. Nevertheless, there are many concerns with the use of MTBF and Appendix 1 discusses some of the problems introduced by selecting a common metric at the system level.

Recommendation 3. A global metric should be developed and validated that can address the reliability of both hardware and software and provide a meaningful connotation at the system level.

To avoid difficulties in marrying hardware and software measurements, a more general definition may be used. "Reliability for a system should be stated as (1) the per cent reliability required and (2) the mission time, instead of condensing these two numbers into one number (MTBF) " [1]. The time basis for the system will be mission elapsed time, incorporating operating time for the hardware and execution time for the software. For this factor, operator performance has not been included—still a broader context might consider mission performance which would not remove the human element from the overall assessment. For systems with large software components, other factors need to be addressed in order to show how reliability objectives are affected by software as a function of operational exposure and maintenance effort. These factors include operating mode, failure severity, maintenance action, fault tolerance provisions, fault density, and software reliability growth. [We expect these issues to be addressed at the software level by the product and process panels and then reflected back to the system level in terms of a reliability metric].

Recommendation 4. Additional research is needed in order to establish the relationship between software metrics (such as complexity, test effort, fault density and process maturity) and percent reliability and to determine correct composite system reliability metrics (incorporating both software metrics and hardware reliability figures of merit).

Currently, system reliability requirements, in the form of MTBF or failure rate, are specified up front for the hardware in systems. The reliability requirements are derived from and based upon mission and user requirements.

Recommendation 5. Acquisition Guidelines should include preliminary definitions of metrics for software reliability, maintainability and other parameters suitable for combining with hardware metrics.

Techniques and procedures exist which enable the hardware developer to measure, monitor and control the reliability of his product throughout the various stages of the design and development process. Part and assembly characteristics, operating time and the environmental stresses (temperature, vibration, etc.) to which the hardware will be exposed during mission performance are some of the factors which must be taken into account in assessing hardware reliability. Given that a reliability requirement is levied against a system containing both hardware and software, then similar, but of course not identical, procedures could be used for assessing reliability and for monitoring and controlling the software.

Recommendation 6. Software error (and reliability) budgets should be assigned to major software subsystems during system design, just as weight, power, reliability, etc. budgets are now assigned to hardware functions. Techniques to predict and measure software reliability should be vigorously pursued.

MTBF or failure rates for software, although conceptually similar to that for hardware, need to be estimated based upon the software execution time, and the residual fault content as well as other factors related to software characteristics. Methods of acceptance testing must also be devised which permit demonstration of the achievement of reliability objectives. The important point is that HW/SW metrics be compatible so that they can be combined to obtain a reliability measure for the total system. Metrics such as error/fault density which are directly related to the software reliability should be monitored and controlled during system development. Test techniques for performing such monitoring are also needed.

Recommendation 7. A software reliability metric commensurate with hardware MTBF is needed. This might require (or at least, benefit from) changes in the ways software modules are defined. Acceptance tests should demonstrate achieved reliability in terms of such a metric.

3.5.4.2 Maintainability

Maintainability corresponds to the maximum tolerable time that a system would be out of service due to a system (hardware, software, or correlated) failure. This factor is characterized by several hardware component and software attributes and can be measured in terms of mean time to repair (MTTR).

The MTTR (Mean Time To Repair) is specified up front for the system's hardware. A similar measure for software can also be used when system availability is of critical concern. However, maintenance practices for hardware versus that for software represent substantially different scenarios. For example, the need for hardware maintenance sets up logistic requirements for spare parts, on-site repair personnel at intermediate and depot levels, as well as large investments in test equipment and maintenance facilities. Software maintenance, however, is largely concerned with redesign and providing updates or extended capability for the system. Correspondingly, different definitions and and interpretation of the MTTR measure must be made. Again, the important point is that compatible measures for hardware and software be used so that a system oriented measure can be obtained.

Recommendation 8. When tracking maintenance activity, distinctions should be made between fixing software errors that have caused failures and system improvements or enhancements.

The system user is primarily concerned with having the system available for use when needed to perform the mission. System metrics such as MTBF and MTTR, when properly specified, monitored and controlled throughout the development process, will provide the user with a means to verify achievement of availability and mission requirements. Metrics not related to system quality, mission performance or reliability requirements are, in general, of secondary concern to the customer.

3.5.4.3 Digital Performance

Digital Performance addresses the rate at which computer supported functions can be carried out and is a product of processing rate and algorithmic efficiency. Measures of digital performance should help relate predicted performance, as characterized by a conceptual (prototype) system performance model, to measured performance of the real system. A performance model may be built up from several application benchmarks with well known performance characteristics, Monte Carlo simulations, or by heuristics. Execution time metrics that would compare the model's expected time to complete with actual execution time would be most appropriate here.

The panel agreed that system metrics with regard to performance of the hardware-software in an application may best be represented by a set of benchmarks built around a combination of critical functions implemented in software. The critical functions selected would ideally incorporate the algorithms which will be the most stressing with regard to schedule deadlines, internal table/storage capacity, etc. The benchmarks would consist of the software capturing the critical functions, and a set of program drivers or scenarios to exercise the software.

Recommendation 9. Investigate the use of a model/benchmarks to capture the interrelationships of the hardware and software with regard to performance and the other "ilities."

The benchmarks would be used to:

(a) establish the execution bounds the critical functions must live within.
(b) establish, together with the execution bounds, the execution degradation factor of the conceptual model, i.e. establish the baseline performance of the conceptual model.
(c) help decide the hardware/software allocation.within the execution bounds constraints.
(d) assist in evaluation of hardware-compiler combinations as candidates for the development and target hardware/software configurations.

3.5.4.4 Extendibility

Extendibility corresponds to the available extra capacity for change or flexibility built into the system that allows easy adaptation to new requirements or extensions/changes to existing requirements. This factor is best described in attributes of additional processing time available, excess allocated memory, excess pc board real estate, etc, but can also be characterized in terms of man-hours necessary to make the change or adaptation.

3.5.4.5 Safety

System safety is best characterized by the identification of system events that can be tied to a failure that could be harmful to people, equipment, or the system itself. Techniques like failure mode effects analysis (FMEA) should be part of the requirements definition process. From this baseline of all events that could occur, a percentage of events that effect safety could be identified and tracked.

3.5.4.6 Usability

Usability corresponds to the user's ability to effectively use the system as it was intended. This factor may be characterized by several ergonomic/human factors attributes related to application workload like usage fatigue, training time, and disorientation effects. These attributes are generally measurable in time units.

The panel agreed that system metrics with regard to performance of the hardware-software in an application may best be represented by a set of benchmarks to provide a partial assessment of the system's usability.

3.5.5 Conceptual Model for System Metrics

To describe a system perspective for hardware and software metrics, we created a conceptual model for explaining the processes we wished to describe in the context of a development life cycle but our model is not itself a life cycle. Mission requirements (in addition to functional requirements) were considered the starting point (see Figure 1).

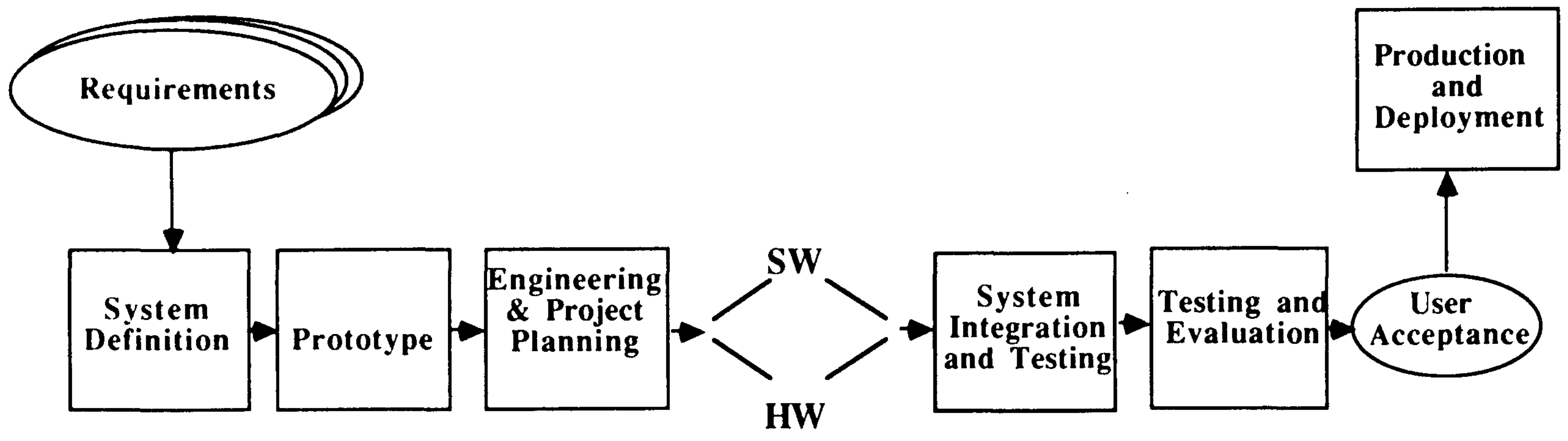
Recommendation 10. *Projects should identify the system's* quality requirements to complete a given mission—including measurable attributes of reliability, maintainability, digital performance, extendibility, safety and usability.

3.5.5.1 System Definition/System Architecture in Preliminary Design

From a metric standpoint, as well as for a better understanding of the system, vague or imprecise requirements may need to be made more precise and quantitative during system definition. For example, a user may want an aircraft door that is easy to open and close. Such a vague requirement is not sufficient.; it may need to be refined in stages until it can be expressed more quantitatively (e.g., no more than 3 foot-pounds of force required to open or close the door).

Recommendation 11. Total Quality Management (TQM) techniques, such as Quality Function Deployment (QFD), should be investigated for use in translating customer and mission requirements into software specifications and ultimately into specific design and reliability requirements.

DoD-wide, experience has taught that "up-front" system work has exceptionally large leverage vs similar work carried out later in the acquisition cycle. This early activity traditionally includes definition of a system concept, and decomposition (partitioning) of the system into functional or physical subsystems. However, there is no universal understanding as to what the functional activities should include or how detailed they


Requirements
System Definition
Prototype
Engineering & Project Planning
SW
HW
System Integration and Testing
Testing and Evaluation
User Acceptance
Production and Deployment

Reliability	Per Cent Reliability and Mission Time
Maintainability	MTTR
Digital Performance	Application Benchmarks
Extendibility	Man-Hours
Safety	Per Cent of Failures that affect Safety Events
Usability	Application Workload

Figure 1. Conceptual Model for System Metrics

should be. Even at this early point in the cycle, the desire is to establish target values for mission capability in terms of metrics—in fact, sometimes a project needs to both define metrics and assign objective *metric values*. Such definitions often require detailed system characterization down to functional subsystem levels.

Traditionally, system concepts have not characterized software and hardware in a balanced manner. It might have been argued that this was not necessary, since the system designer's point of view often was that '...software will be written to perform all necessary system functions'. Software often was asked to implement requirements neglected in the initial design and hence absent from the hardware architecture. This limited characterization created problems in thinking about the project from a total system's perspective.

Now, with increased awareness of software characteristics, we are ready to begin avoiding such limitations by seriously considering allocating system budgets (for example, errors) separately to software and hardware elements. This requires that combined software and hardware architectures be developed. One advantage in doing this is that "software functions" can thus more sensibly be allocated between several different processors, or even performed in non-programmed hardware. The universality of small, one or few-chip processors makes this flexible allocation more attractive than ever before.

In addition to error budgeting, functional performance budgeting should be dealt with prior to detailed subsystem definition. If, for example, only 10 milliseconds were allowable in total delay through some set of complex software-complemented functions, preliminary consideration of precision required and need for parallel processing should be made. Though these trades could be revisited, if the software combination were assigned to one system designer, the issue of realizability can be settled at an architectural level—rather than much later, when the software is under development.

Recommendation 12. TQM techniques, such as Statistical Process Control (SPC) for monitoring and controlling specific product/process metrics should be investigated for use in the software development environment. In addition, quality management activities which focus on "causal analysis" (i.e. determining the root cause of software errors/faults and eliminating them from the process or product through corrective action) should also be fully promoted and investigated further, if necessary.

Recommendation 13. Software module definition methods which can reduce likelihood of failure due to module-to-module incompatibilities should be studied and results applied at system definition.

Recommendation 14. Real-time systems, distributed systems and parallel processing systems need to be supported with additional research to determine the appropriate system definition tasks that should be conducted.

3.5.5.2 Conceptual Prototype Used Throughout Acquisition

A conceptual prototype system is defined to be a model system of hardware and software which, when stimulated by simulated generational and environmental inputs, performs all (or perhaps the most critical) system operations, responding to stimuli and user controls just as would the actual system. However, it may differ from the actual system in using different processor(s), programming language(s) and other technology, and not operating in the actual land, aerospace, or marine system environment.

Such a prototype is an excellent vehicle for use in (1) verifying system design, (2) locating design errors, and (3) evaluating and refining user interfaces. Part of the merit of a prototype lies in the use of powerful, easy to change programming languages which permit rapid changes of system functioning and easy comparison of alternative approaches.

Such prototypes are not new in concept. However, while traditional employment of prototypes is limited to early phases of system development, we propose use of extended-life system prototypes through the system development integration, and initial operational testing cycles. The cost of keeping the prototype working should be modest but returns on such an investment could be very great. Such an extended life prototype could provide a basis for testing the actual system's operation. Any observed exception in system operation can be compared against the operation of the prototype to verify whether an exception is due to design or to implementation of the software. In addition, most components of the prototype system (hardware as well as software) can be reused, once the actual system has replaced the prototype, as the vehicle for exploring system change and enhancement after deployment.

Prototypes are useful for helping developers and users to determine if their functional requirements are understood and complete. We would like to see the prototypes used to show whether quality or system requirements are also understood. Continuing our earlier example, a prototype aircraft door may be built to determine how much force is actually used to operate such a door—thereby learning much earlier that 3 foot-pounds does not really make it easy to open and close for the average person and perhaps the requirement should be changed to no more than 2 foot-pounds. Conceptual prototypes should exist until the point in system integration that a more valid replacement in terms of a functioning actual system is available. It would be useful to match behavior in the prototype to that being achieved in the actual system. For example, key algorithms or benchmarks expressed in the prototype could help determine if performance for those functions are being met. Similarly, errors found in the actual system can be used to refine the conceptual prototype as well.

Recommendation 15. Prototypes should be used throughout the life cycle to demonstrate quality requirements as well as functional requirements and to explain system changes and enhancements after deployment.

3.5.5.3 Engineering and Project Planning / Hardware and Software Functional Allocation

A key part of engineering and project planning is performing hardware and software functional allocation. One objective of architectural definition is to identify necessary interfaces and select interface implementations (bus or other). A second objective is the preliminary definition of subsystem functional criteria. A third is the demonstration that required values of system metrics can (in principle, at least) be realized by the allocation of performance, reliability and other characteristics between subsystems. When a single processor represents a set of subsystems (e.g. via time-sharing), it must satisfy metrics requirements in a collective manner for all subsystems.

It is essential that the fundamental allocations of function and of metrics be made early in system design, even though subject to corrections during later phases. Even where all software functions will be implemented in a single processor, it is not sufficient to merely assure that inputs and outputs are available, and guess that processor power and memory capacity will be "more than enough." Function should be defined and dimensioned, critical algorithms identified, and realizability verified (through modeling or prototypes, if need be).

Allocation of the value of a system metric between hardware and software subsystems depends on the nature of the metric. While latency (time) is the sum of latencies of cascaded functions, mean time between failures (MTBF) usually combines inversely in hardware: the combined MTBF of a system comprising subsystem 1 and 2 is usually given by

$$\frac{1}{MTBF_{system}} = \frac{1}{MTBF_1} + \frac{1}{MTBF_2}$$

This assumes independence between functional elements.

The equivalent MTBF of two connected software functions 1 and 2, however, will more accurately be described as

$$\frac{1}{MTBF_{system}} = \frac{1}{MTBF_1} + \frac{1}{MTBF_2} + \frac{1}{MTBF_{12}}$$

where the final term represents the situation where 1 and 2 have incompatibilities which result in system error. Clearly if the $MTBF_{12}$ term can be made extremely <u>large</u>, the problem of software error rates must be helped.

Recommendation 16. Software module characteristics which reduce chances of failure by cooperative error between modules should be better understood.

3.5.5.4 System Integration and Testing

Test plans for all testing activity should be laid out when system decomposition is determined, but System Integration and Testing falls under the domain of the System Project Office (SPO). Unit testing (i.e., testing each unit or hardware component individually) has already been conducted and is not part of the system activity; it is still considered as belonging to either hardware or software development. While software test measures (e.g., test effort, test coverage) at the unit or component level need to be examined at a detailed level, as a project moves through System Integration and Testing, measurements should gradually become more devoted to the system factors, e.g., reliability expressed as per cent reliability achieved on actual tests. Testing at this level concentrates on showing that hardware components and/or software units can meet their specification, but the knowledge gained from the test process can still be used to help determine whether system requirements are going to be met.

3.5.5.5 Testing and Evaluation

Testing and Evaluation usually falls under the domain of an outside organization (e.g., AFOTEC for the Air Force). At this level of testing, <u>validation</u> becomes more critical; i.e., showing that the assembly of hardware components and software units can function properly in a total systems environment. Demonstration tests for reliability should become part of the focus, using random inputs or, if possible, actual input data collected randomly from sampling the operational environment. By the conclusion of Testing and Evaluation, results from the demonstration tests should show whether required system factors have been met.

3.5.5.6 User Acceptance

User acceptance is a formal process, consisting of tests to determine whether the developed system meets predetermined functionality, performance and quality.

Recommendation 17. Incentives for fulfilling the requirements for system factors should be part of user acceptance and acquisition managers must program for such incentives.

Recommendation 18. The use of Reliability Warranties/Guarantees for software should be investigated. Warranties, as a contracting and reliability assurance vehicle, have been successful on hardware programs when properly planned and applied. Software may lend itself to warranty application even more readily than hardware.

3.5.5.7 Deployment

During deployment, the final proof of whether system factor requirements were truly met can be determined. System maintenance also begins and encompasses all activities required to ensure that the system continues to function properly, even as the original problem and environment change.

Recommendation 19. Sufficient data should be collected during operational use to provide meaningful determination of the level achieved for specified system factors and to prepare cause and effect diagrams. Causal analyses should be conducted to determine the cause of any critical failure or any critical change to the system and used as part of a feedback loop to developers to achieve quality improvement.

3.5.6 Summary: Recommendations for System Metrics

1. System quality requirements for reliability, maintainability, digital performance, extendibility, safety, and usability should be identified in the Request for Proposal (RFP) and should reflect the customer's inputs and concerns.

2. Further research should be conducted to determine a complete set of system quality factors and appropriate measures for them.

3. A global metric should be developed and validated that addresses the reliability of both hardware and software and is meaningful at the system level.

4. Additional research should be started to relate detailed software metrics such as complexity, test effort or technique, fault density and maturity to percent reliability and to combine software metrics with hardware reliability figures of merit..

5. Acquisition Guidelines should include preliminary definitions of high level metrics for software reliability, maintainability and other parameters suitable for combining with hardware metrics.

6. Software error (and per cent reliability) budgets should be assigned to major software subsystems during system design, just as weight, power, error, etc. budgets are now assigned to hardware functions. Techniques to predict and measure software reliability across the life cycle should be vigorously pursued.

7. A software reliability metric similar to hardware MTBF is needed. This might require (or at lease, benefit from) changes in the ways software modules are defined. Acceptance tests should demonstrate achieved reliability in terms of such a metric.

8. Distinctions should be made when tracking maintenance activity between fixing software errors that have caused failures and system improvements or enhancements.

9. Investigate the use of a model/benchmarks to capture the interrelationships of the hardware and software with regard to performance and the other "ilities." One possible approach is to capitalize on the power of tools such as logic programming and expert systems shells which can capture relationships and allow the creation of "theorems" based on those relationships. These "theorems" could be checked and refined throughout the development cycle to assure the decisions made and the performance achieved in the hardware and software are consistent with the system requirements.

10. Projects should identify the system's quality requirements to complete a given mission—including measurable attributes of reliability, maintainability, digital performance, extendibility, safety and usability.

11. Total Quality Management (TQM) techniques, such as Quality Function Deployment (QFD) should be investigated for use in translating customer and mission requirements into software specifications and ultimately into specific design and reliability requirements.

12. TQM techniques, such as Statistical Process Control (SPC) for monitoring and controlling specific product/process metrics be investigated for use in the software development environment. *In addition, quality management activities which focus on* "causal analysis" (i.e. determining the root cause of software errors/faults and eliminating them from the process or product through corrective action) should also be fully promoted and investigated further, if necessary.

13. Software module definition methods which can reduce likelihood of failure due to module to module incompatibilities should be studied and results applied at system definition.

14. For real-time systems, distributed systems and parallel processing systems, there is a much closer connection between the software and the hardware environment than in traditional systems. Analysts and designers need to be supported with additional research to determine the appropriate system definition tasks that should be conducted for such systems.

15. Prototypes should be used throughout the life cycle to demonstrate quality requirements as well as functional requirements and to explain system changes and enhancements after deployment.

16. Software module characteristics which reduce chances of failure due to cooperative errors between modules need to be studied further.

17. Incentives for fulfilling the requirements for system quality factors should be part of user acceptance.

18. The use of Reliability Warranties/Guarantees for software should be investigated. Warranties, as a contracting and reliability assurance vehicle, have been successful on hardware programs when properly planned and applied. Software may lend itself to warranty application more readily than hardware. As an example, one business company, Du Pont Informational Engineering Associates, offers fixed price solutions for custom software for critical business applications with a money—back guarantee.

19. Sufficient data should be collected during operational use to provide meaningful determination of the level achieved for specified system factors and to prepare cause and effect diagrams. Causal analyses should be conducted to determine the cause of any critical failure or any critical change to the system and used as part of a feedback loop to developers to achieve quality improvement.

20. The utility of system definition languages for unambiguous and complete definition of function implemented in software should be studied, using existing languages, if applicable. (Reference Appendix II)

21. Very high level languages (procedural and declarative) should be developed and used to provide ways to define system functions. (Reference Appendix II)

3.5.7 References

1. J. M. Juran, Quality Control Handbook, Fourth Edition, McGraw Hill,1988.

3.5.8 Appendix I

MTBF as a Software Metric: A Negative View

MTBF is used almost universally as a system metric. One must recognize that it requires clear definition: vis, a real system may have a number of partially failed states which yield partial system operation. However, in practice this has posed no serious limitation to use of MTBF as a system metric.

Hardware failures may be intermittent or permanent or. Any given error may leave a result which ranges from no system impact to system destruction. In MTBF estimates, MTBF of a system is determined by combining MTBF for components. Components are described as having a given MTBF even though in some cases there is far more uncertainty $(\mathrm{MTBF})^2$ than in others. In any event, the approach has proved a satisfactory one for achieving well-balanced system designs.

Software errors may occur when the system is in some given state and experiences a combination of inputs for which the option was not designed or was not tested. As in the case of hardware errors, the fault may not produce a detected error. The result may range from continued correct system operation to system destruction. Treating software errors via an MTBF model is equivalent to assuming that inputs and states during operation pass randomly and unpredictable through all combinations of physically or logically realizable values. If a software error is detected, located, and repaired, that error should not recur in the system. Hence, if no new errors are introduced by the repair, the MTBF should increase, as errors in the original design are removed. In practice this is unachievable, since repair procedures are error-prone. With this understanding that, in fact, the errors are random only if inputs are, then MTBF may be a useful (though gross) measure of the software elements of a system.

Unfortunately, the principal combinational technique which makes MTBF useful does not apply well to software. MTBF of a system containing M components whose MTBFs are $T_1, T_2, \ldots, T_M$, when failure of any one component causes system failure, can be found as

$$\mathrm{MTBF}_{\text{system}} = \frac{1}{\frac{1}{T_1} + \frac{1}{T_2} + \ldots + \frac{1}{T_M}}$$

The result depends upon the <u>independence</u> of failures of the components.

In software, if one views individual instructions as software "components," each instruction has no software failure likelihood; errors occur only because instruction <u>sequences</u> are incorrect. Even if we identify <u>software modules</u> as components, each module may have no errors with respect to performing according to its own specification. However, errors can still occur due to incorrect module specifications, or improper relationship of modules. Accordingly, the powerful aggregation technique used in defining hardware MTBF has no applicability to software. Reconciling this viewpoint with the desire to have a system metric which combines hardware and software issues lead to some of the compromises that the panel made.

3.5.9 Appendix II

System Definition Languages in System Software Definition

It has repeatedly been suggested by proponents of hardware definition languages that describing system (or, more likely, subsystem) function using a representation similar to a programming language provides improved testability. Ultimately, an unambiguous definition of this type might be used to control an automatic program generator, while itself providing a basis for compliances and/or correctness evaluations.

Recommendation 20. The utility of system definition languages for unambiguous and complete definition of function implemented in software should be studied, using existing languages, if applicable.

A VHSIC Hardware Development Language (VHDL) based on Ada was developed originally as a vehicle to use for describing semiconductor chip designs. Clearly, very high level languages (procedural and declarative) could provide ways to define many or most system functions. In limited application domains such as database searching, such languages are already in wide use to code applications.

One advantage of a high level language approach, in respect to software quality and metrics is that virtually error-free software could become feasible. In addition, the values of other metrics could be quickly calculated direct from the language description without need to generate and exercise the full system.

Recommendation 21. Very high level languages (procedural and declarative) should be developed and used to provide ways to define system functions.

This page intentionally left blank.

4

BIBLIOGRAPHY

This page intentionally left blank.

4 BIBLIOGRAPHY

For additional information on software metrics and measurement technology, the following material is available:

Conte, Samuel Daniel; Dunsmore, H. E.; Shen, V. Y.; SOFTWARE ENGINEERING METRICS AND MODELS. 396 p. Avail. from Addison Wesley,Benjamin/Cummings Publ. Co., Inc.,Jacob Way, Reading, MA 01867. Order No. ISBN 0-8053-2162-4.

Caswell, Deborah L.; Grady, Robert B.; SOFTWARE METRICS: ESTABLISHING A COMPANY-WIDE PROGRAM. 300 p. Avail. from Prentice-Hall, Rt. 59 at Brook Hill Drive, West Nyack, NY 10995. Order No. ISBN 0-13-821844-7.

Arhire, Romulus; Ivan, Ion; Macesanu, Marian; "PROGRAMS COMPLEXITY: COMPARATIVE ANALYSIS HIERARCHY, CLASSIFICATION," In ACM SIGPLAN Notices. 22(4): Apr 1987. pp. 94-102.

Kafura, Dennis; Reddy, Geereddy R.; "THE USE OF SOFTWARECOMPLEXITY METRICS IN SOFTWARE MAINTENANCE," In IEEE Transactions Software Engineering. 13(3): Mar 1987. pp. 335-343.

Ejiogu, Lem O.; "THE CRITICAL ISSUES OF SOFTWARE METRICS," In ACM SIGPLAN Notices. 22(3): Mar 1987. pp. 59-64. Iannino, Anthony; Musa, John D.; Okumoto, Kazuhira; SOFTWARE RELIABILITY: MEASUREMENT, PREDICTION, APPLICATION. 621 p. Avail. from McGraw-Hill Book Company, Princeton Road, Highstown, NJ 08520. Order No. ISBN 0-07-044093-X.

Sneed, Harry M.; "DATA COVERAGE MEASUREMENT IN PROGRAM TESTING," In Proceedings Workshop on Software Testing. Jul 1986. pp. 34-40. Avail. from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Order No. TH0144-6/0000/0034.

Fairbanks, Karl S., Jr.; Neider, Jeffrey L.; ADAMEASURE: AN ADA SOFTWARE METRIC. 165 p. Mar 1987. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A177 477.

Bowen, C.; Fenwick, S.; Hecht, H.; Hecht, M.; McCall, J.; McKelvey, N.; Morris, John; Randall, W.; Senn, Roy; Vienneau, Robert; Yates, P.; METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION. Vol 1. 210 p. Nov 1987. Alexandria, VA 22304-6145. Order No. AD-A190 018.

Bowen, C.; Fenwick, S.; Hecht, H.; Hecht, M.; McCall, J.; McKelvey, N.; Morris, John; Randall, W.; Senn, Roy; Vienneau, Robert; Yates, P.; METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION. Vol 2. 181 p. Nov 1987. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A190 019.

PROCEEDINGS OF THE 5TH ANNUAL NATIONAL CONFERENCE ON ADA TECHNOLOGY HELD IN ARLINGTON, VIRGINIA ON MARCH 16-19, 1987. 510 p. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A178 690.

Harrison, Warren; "INTRODUCTION TO THE SPECIAL ISSUE ON SOFTWARE ENGINEERING," In Journal of Systems and Software. 8(1): Jan 1988. pp. 1-2.

Baker, Albert L.; Bieman, James M.; Clites, Paul N.; Gustafson, David A.; Melton, Austin C.; "A STANDARD REPRESENTATION OF IMPERATIVE LANGUAGE PROGRAMS FOR DATA

COLLECTION AND SOFTWARE MEASURES SPECIFICATION," In Journal of Systems and Software. 8(1): Jan 1988. pp. 13-37.

Dunsmore, H. E.; Nejmeh, Brian A.; Shen, Vincent Y.; Yu, Tze-Jie; "SMDC: AN INTERACTIVE SOFTWARE METRICS DATA COLLECTION AND ANALYSIS SYSTEM," In Journal of Systems and Software. 8(1): Jan 1988. pp. 39-46.

Harrison, Warren; "MAE: A SYNTACTIC METRIC ANALYSIS ENVIRONMENT," In Journal of Systems and Software. 8(1): Jan 1988. pp. 57-62.

Card, David; MEASURING SOFTWARE DESIGN. Report No. NASA-TM- 89372. 77 p. Nov 1986. Avail. from National Technical Information Service 5285 Port Royal Rd, Springfield, VA 22161. Order No. N87-23189.

Deloach, Scott A.; ENVIRONMENT PORTABILITY AND EXTENSIBILITY MEASURES. Report No. AFIT-EN-TM-87-7. 7 p. Aug 1987. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A184 947.

Cook, Curtis; Harrison, Warren; "A MICRO/MACRO MEASURE OF SOFTWARE COMPLEXITY," In Journal of Systems and Software. 7(3): Sep 1987. pp. 213-219.

Evans, Michael W.; Marciniak, John J.; SOFTWARE QUALITY ASSURANCE & MANAGEMENT. 327 p. Avail. from John Wiley & Sons, Inc., 1 Wiley Drive, Attn: Order Dept., Summerset, NJ 08873. Order No. ISBN 0-471-80930-6.

Sunderhaft, Nancy L.; Vienneau, Robert; STARS MEASUREMENT SURVEY SUMMARY. 162 p. May 1986. Avail. from Data & Analysis Center for Software, P.O. Box 120, Utica, NY 13503.

Data & Analysis Center for Software; THE DACS MEASUREMENT ANNOTATED BIBLIOGRAPHY: A BILIOGRAPHY OF SOFTWARE MEASUREMENT LITERATURE (MAY 1986). Report No. MBIB-1. 263 p. May 1986. VA 22304-6145. Avail. from Data & Analysis Center for Software, P.O. Box 120, Utica, NY 13503.

Hsueh, Mei-Chen; MEASUREMENT-BASED RELIABILITY/ PERFORMABILITY MODELS. Report No. UILU-ENG-87-2258. 111 p. Sep 1987. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A185 203.

IEEE GUIDE FOR THE USE OF STANDARD DICTIONARY OF MEASURES TO PRODUCE RELIABLE SOFTWARE. Report No. IEEE Std 982.2-1988. 214 p. May 1988. Avail. from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854.

Jones, Capers; PROGRAMMING PRODUCTIVITY. 292 p. Avail. from McGraw-Hill Book Company, Princeton Road, Highstown, NJ 08520. Order No. ISBN 0-07-032811-0.

Bourque, P.; Cote, Vianney; Oligny, S.; Rivard, N.; "SOFTWARE METRICS: AN OVERVIEW OF RECENT RESULTS," In Journal of Systems and Software. 8(2): Mar 1988. pp. 121-131.

National Technical Information Service; CITATIONS FROM THE INFORMATION SERVICES FOR THE PHYSICS AND ENGINEERING COMMUNITIES INSPEC DATABASE: SOFTWARE QUALITY AND METRICS (JAN 87-DEC 87). 49 p. Jan 1988. Avail. from National Technical Information Service 5285 Port Royal Rd, Springfield, VA 22161. Order No. PB88-856703.

COMPSAC 86, PROCEEDINGS. Report No. IEEE 86CH2356-4. 504 p. Avail. from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Order No. ISBN 0-8186 0727-0.

Navlakha, Jainendra K.; "MINIREVIEW OF SOFTWARE PRODUCTIVITY METRICS," In COMPSAC 1986, Proceedings. Oct 1986. pp. 4-5.

Azuma, Motoei; Cavano, Joseph; Hecht, Herbert; McKissick, Jack; Yasuda, Katsuyuki; "PANEL - SOFTWARE QUALITY MANAGEMENT: USA AND JAPANESE PERSPECTIVES," In COMPSAC 1986, Proceedings. Oct 1986. pp. 87-89.

Cerino, Deborah A.; "SOFTWARE QUALITY MEASUREMENT TOOLS AND TECHNIQUES," In COMPSAC 1986, Proceedings. Oct 1986. pp. 160- 167.

Lopez, M. A.; Rodriguez, V.; Tsai, W. T.; Volovik, D.; "AN APPROACH TO MEASURING DATA STRUCTURE COMPLEXITY," In COMPSAC 1986, Proceedings. Oct 1986. pp. 240-246.

Shatz, S. M.; "ON COMPLEXITY METRICS ORIENTED FOR DISTRIBUTED PROGRAMS USING ADA TASKING," In COMPSAC 1986, Proceedings. Oct 1986. pp. 247-253.

Melton, Austin; Ramamurthy, Bina; "A SYNTHESIS OF SOFTWARE SCIENCE METRICS AND THE CYCLOMATIC NUMBER," In COMPSAC 1986, Proceedings. Oct 1986. pp. 308-313.

Levitin, Anany V.; HOW TO MEASURE SOFTWARE SIZE, AND HOW NOT TO. In COMPSAC 1986, Proceedings. pp. 314-318.

Knafl, George J.; Sacks, Jerome; "SOFTWARE DEVELOPMENT EFFORT PREDICTION BASED ON FUNCTION POINTS," In COMPSAC 1986, Proceedings. Oct 1986. pp. 319-325.

Rodriguez, Volney; Tsai, Wei-Tek: "SOFTWARE METRICS INTERPRETATION THROUGH EXPERIMENTATION," In COMPSAC 1986, Proceedings. Oct 1986. pp. 368-374.

Longworth, Herbert D.; Ottenstein, Linda M.; Smith, Martyn R.; "THE RELATIONSHIP BETWEEN PROGRAM COMPLEXITY AND SLICE COMPLEXITY DURING DEBUGGING TASKS," In COMPSAC 1986, Proceedings. Oct 1986. pp. 383-389.

Osborne, Wilma; "MINIREVIEW - SOFTWARE MAINTENANCE," In COMPSAC 1986, Proceedings. Oct 1986. pp. 391-402.

9TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. Report No. IEEE Catalog 87CH2432-3. 415 p. Avail. from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Order No. ISBN 0-89791-216-0.

Miyazaki, Yukio; Murakami, Noritoshi; "SOFTWARE METRICS USING DEVIATION VALUE," In 9th International Conference on Software Engineering. Mar 1987. pp. 83-91.

Basili, Victor R.; Rombach, H. Dieter; "TAME -TAILORING A MEASUREMENT ENVIRONMENT," In Proceedings of the 11th Annual Software Engineering Workshop: Dec 1986. Dec 1986. 24 p.

Gibbons, John J.; Hall, David; Knell, Marlene; "QUANTIFYING PRODUCTIVITY: A SOFTWARE METRICS DATA BASE TO SUPPORT REALISTIC COST ESTIMATION," In Proceedings of the International Society of Parametric Analysts 8th Annual Conference 1986. May 1986. pp. 496-513.

Branyan, E.; Rambo, R.; Thiel, L.; "ESTABLISHMENT AND VALIDATION OF SOFTWARE METRIC FACTORS," In Proceedings of the International Society of Parametric Analysts 8th Annual Conference 1986. May 1986. pp. 562-592.

Ivanek, B.; Kokol, P.; Zumer, V.; "SOFTWARE EFFORT METRICS: HOW TO JOIN THEM," In Software Engineering Notes (ACM SIGSOFT). 13(2): Apr 1988. pp. 55-57.

Duncan, Anne Smith; "SOFTWARE DEVELOPMENT PRODUCTIVITY TOOLS AND METRICS," In 10th International Conference on Software Engineering: April 11-15, 1988. Apr 1988. pp. 41-48.

Benjamin, James L.; "PILOT: A PRESCRIPTION FOR PROGRAM PERFORMANCE MEASUREMENT," In 10th International Conference on Software Engineering: April 11-15, 1988. Apr 1988. pp. 388-395.

Boehm, Barry W.; "IMPROVING SOFTWARE PRODUCTIVITY," In Tutorial: Software Engineering Project Management. Jan 1988. pp. 93-107.

PROCEEDINGS OF THE 1988 ACM SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS. 293 p. Avail. from ACM Order Department, P. O. Box 64145, Baltimore, MD. 21264. Order No. ISBN 0-89791-254-3.

Haban, Dieter; Wybranietz, Dieter; "MONITORING AND PERFORMANCE MEASURING DISTRIBUTED SYSTEMS DURING OPERATION," In ACM SIGMETRICS - Performance Evaluation Review. Volume 16, No. 1. May 1988. pp. 197-206.

Koss, W. Edward; "SOFTWARE - RELIABILITY METRICS FOR MILITARY SYSTEMS," In 1988 Annual Reliability and Maintainability Symposium. Jan 1988. pp. 190-194.

Hess, James A.; "MEASURING SOFTWARE FOR ITS REUSE POTENTIAL," In 1988 Annual Reliability and Maintainability Symposium. Jan 1988. pp. 202-207.

Hartley, Richard; Pierce, Patricia; Worrells, Suellen; SOFTWARE QUALITY MEASUREMENT DEMONSTRATION PROJECT II. Report No. RADC-TR-87-164. 156 p. Oct 1987. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A192 849.

Weyuker, Elaine J.; "EVALUATING SOFTWARE COMPLEXITY MEASURES," In IEEE Transactions on Software Engineering. 14(9): Sep 1988. pp. 1357-1365.

Boehm, Barry W.; Papaccio, Philip N.; "UNDERSTANDING AND CONTROLLING SOFTWARE COSTS," In IEEE Transactions on Software Engineering. 14(10): Oct 1988. pp. 1462-1477.

Delaney, Richard P.; Summerill, Lawrence F.; "ADA SOFTWARE METRICS," In Proceedings of Sixth National Conference on Ada Technology. Mar 1988. pp. 24-31.

Anderson, Jane D.; Perkins, John A.; "EXPERIENCE USING AN AUTOMATED METRICS FRAMEWORK IN THE REVIEW OF ADA SOURCE FOR WIS," In Proceedings of Sixth National Conference on Ada Technology. Mar 1988. pp. 32-41.

Poore, J. H.; "DERIVATION OF LOCAL SOFTWARE QUALITY METRICS (SOFTWARE QUALITY CIRCLES)," In Software -Practice and Experience.18(11): Nov 1988. pp. 1017-1027.

Knudson, Richard W.; Reifer, Donald J.; Smith, Jerry; FINAL REPORT: SOFTWARE QUALITY SURVEY. Report No. RCI-TR-057. 48 p. Nov 1987. Avail. from Reifer Consultants, Incorporated, P.O. Box 4046, Torrance, CA.

Reifer Consultants, Inc.; PRODUCTIVITY, QUALITY AND METRICS. Report No. RCI-TN-303. 43 p. Oct 1987. Avail. from Reifer Consultants, Incorporated, P.O. Box 4046, Torrance, CA.

Schultz, H. P.; SOFTWARE MANAGEMENT METRICS. Report No. ESD-TR-88-001. 52 p. May 1988. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A196 916.

McManus, James I.; Schulmeyer, G. Gordon; HANDBOOK OF SOFTWARE QUALITY ASSURANCE. 480 p. Avail. from Van Nostrand Reinhold Co., 135 W. 50TH ST., NY, NY 10020. Order No. ISBN 0-442-28015-7.

Daughtrey, Taz; "THE SEARCH FOR SOFTWARE QUALITY," In Quality Progress. 21(11): Nov 1988. pp. 29-31.

Murine, Gerald E.; "INTEGRATING SOFTWARE QUALITY METRICS WITH SOFTWARE QA," In Quality Progress. 21(11): Nov 1988. pp. 38-43.

Caro, Israel I.; Higuera, Ronald; Jacobson, Sherwin J.; Kockler, Frank R.; Roberts, Alan; MISSION CRITICAL COMPUTER RESOURCES MANAGEMENT GUIDE: TECHNICAL MANAGEMENT. 222 p. Sep 1988. Avail. from Superintendent of Documents, Government Printing Office, Washington, DC 20402. Order No. S/N 008-020-01152-5.

Johnson, Allen M., Jr.; Malek, Miroslaw; "SURVEY OF SOFTWARE TOOLS FOR EVALUATING RELIABILITY, AVAILABILITY, AND SERVICEABILITY," In ACM Computing Surveys. 20(4): Dec 1988. pp. 227-269.

Fujii, Mamoru; Kudo, Hideo; Sugiyama, Yugi; Torii, Koji; "QUANTIFYING A DESIGN PROCESS BASED ON EXPERIMENTS," In Journal of Systems and Software. 9(2): Feb 1989. pp. 129-136.

McGarry, Frank E.; Valett, Jon D.; "A SUMMARY OF SOFTWARE MEASUREMENT EXPERIENCES IN THE SOFTWARE ENGINEERING LABORATORY," In Journal of Systems and Software. 9(2): Feb 1989. pp. 137-148.

Agresti, William W.; McGarry, Frank E.; "MEASURING ADA FOR SOFTWARE DEVELOPMENT IN THE SOFTWARE ENGINEERING LABORATORY," In Journal of Systems and Software. 9(2): Feb 1989. pp. 149-159.

Collofello, James S.; Woodfield, Scott N.; "EVALUATING THE EFFECTIVENESS OF RELIABILITY-ASSURANCE TECHNIQUES," In Journal of Systems and Software. 9(3): Mar 1989. pp. 191-195.

Gibson, Virginia R.; Senn, James A.; "SYSTEM STRUCTURE AND SOFTWARE MAINTENANCE PERFORMANCE," In Communications of the ACM. 32(3): Mar 1989. pp. 347-358.

Johnson, Stephen K.; MODIFYING AFOTEC'S (AIR FORCE OPERATIONAL TEST AND EVALUATION CENTER'S) SOFTWARE MAINTAINABILITY EVALUATION GUIDELINES. Report No. AFIT/GCS/ENG/88D-10. 123 p. Dec 1988. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A203 381.

Tindell, Douglas R.; MAINTENANCE METRICS FOR JOVIAL (J73) SOFTWARE. Report No. AFIT/GE/ENG/88D-57. 128 p. Dec 1988. Avail. from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145. Order No. AD-A202 713.

Headquarters, Air Force Systems Command (AFSC); ACQUISITION MANAGEMENT: SOFTWARE QUALITY INDICATORS. Report No. AFSCP 800-14. 41 p. Jan 1987. Avail. from HQ, AFSC/XRF, Andrews AFB, Washington, DC 20334.

Headquarters, Air Force Systems Command (AFSC), Andrews AFB, MD 20334; AIR FORCE SYSTEMS COMMAND SOFTWARE MANAGEMENT INDICATORS. Report No. AFSCP 800-43. 21 p. Jun 1988. Avail. from HQ, AFSC/XRF, Andrews AFB, Washington, DC 20334.

Lind, Randy K.; Vairavan, K.; "AN EXPERIMENTAL INVESTIGATION OF SOFTWARE METRICS AND THEIR RELATIONSHIP TO SOFTWARE DEVELOPMENT EFFORT," In IEEE Transactions on Software Engineering. 15(5): May 1989. pp. 649-653.

Jones, C.; "METRIC WITH MUSCLE," In System Development. 9(8): Aug 1989. pp.1-3.

Bollmann, Peter; Zuse, Horst; "SOFTWARE METRICS: USING MEASUREMENT THEORY TO DESCRIBE THE PROPERTIES AND SCALES OF STATIC SOFTWARE COMPLEXITY METRICS," In ACM SIGPLAN Notices. 24(8): Aug 1989. pp. 23-33.

Myers, Margaret; "STRUCTURAL MODELLING OF PROLOG FOR METRICATION," In 2nd European Software Engineering Conference, Univ of Warwick, Sept 11-15, 1989. Sep 1989. pp. 351-375.

Kemerer, Chris F.; "AN AGENDA FOR RESEARCH IN THE MANAGERIAL EVALUATION OF COMPUTER-AIDED SOFTWARE ENGINEERING (CASE) TOOLS IMPACTS," In Proceedings of the 22nd Annual Hawaii International Conference on System Sciences. Jan 1989. pp. 219-228. Order No. 0073-1129/89/0000/0219$01.00.

Chappell, S. G.; Fuchs, E.; "SOFTWARE QUALITY & PRODUCTIVITY AT AT&T BELL LABORATORIES," In Proceedings of the 22nd Annual Hawaii International Conference on System Sciences. Jan 1989. pp. 330-336. Order No. 0073 1129/89/0000/0330$01.00.

Demarco, Tom; Lister, Tim; "SOFTWARE DEVELOPMENT: STATE OF THE ART VS. STATE OF THE PRACTICE," In Proceedings of the 11th International Conference on Software Engineering, May 15-18, 1989. May 1989. pp. 271-275.

Dogsa, Tomaz; Gyorkos, Jozsef; Rozman, Ivan; Welzer, Tatjana; "THE QUALITY OF SOFTWARE - A SURVEY AND ASSESSMENT OF EXISTING RESEARCH WORKS," In 3rd Proceedings of ISCIS III the 3rd International Symposium on Computer and Information Sciences. pp 191-200.

Boehm, Barry; Royce, Walker; "ADA COCOMO AND THE ADA PROCESS MODEL," In The Fifth International COCOMO Users' Group Meeting, SEI October 17-19, 1989. Oct 1989. p. 34.

Klim, S.; "OVERVIEW OF COQUAMO, A CONSTRUCTIVE QUALITY MODEL," In The Fifth International COCOMO Users' Group Meeting, SEI October 17-19, 1989. Oct 1989. p.18.

McGarry, Frank E.; Valett, Jon D.; "A SUMMARY OF SOFTWARE MEASUREMENT EXPERIENCES IN THE SOFTWARE ENGINEERING LABORATORY," In Journal of Systems and Software. 9(2): Feb 1989. pp. 137-148.

Coupal, Daniel; Robillard, Pierre N.; "FACTOR ANALYSIS OF SOURCE CODE METRICS," In Journal of Systems and Software. 12(3): Jul 1990. pp. 263-269.

Russell, Meg; "INTERNATIONAL SURVEY OF SOFTWARE MEASURFMENT EDUCATION AND TRAINING," In Journal of Systems and Software. 12(3): Jul 1990. pp. 233-241.

Bush, Martin E.; Fenton, Norman E.; "SOFTWARE MEASUREMENT: A CONCEPTUAL FRAMEWORK," In Journal of Systems and Software. 12(3): Jul 1990. pp. 223-231.

Oman, Paul W.; Cook, Curtis R.; "DESIGN AND CODE TRACEABILITY USING A PDL METRICS TOOL," In Journal of Systems and Software. 12(3): Jul 1990. pp. 189-198. Order No. 0164-1212/90/$350.

Chang, Pao-Sheng; Yau, Stephen S.; "A METRIC OF MODIFIABILITY FOR SOFTWARE MAINTENANCE," pp 374-381.

Henry, Sallie; Wake, Steve; A MODEL BASED ON SOFTWARE QUALITY FACTORS WHICH PREDICTS MAINTAINABILITY. pp 382-387.

Reynolds, Robert G.; Maletic, Jonathan I.; Provin, Stephen E.; Khoshgoftaar, Taghi, M.; Munson, John C.; "PREDICTING SOFTWARE DEVELOPMENT ERRORS USING SOFTWARE COMPLEXITY METRICS," 8(2): Feb 1990. Report No. IEEE Log Number 8932737. pp.253-261. Order No. 0733-8716/90/0200-0253$01.00.

Gall, Gilbert Le; Adam, Marie-Francoise; Derriennic, Henri; Moreau, Bernard; Valette, Nicole; "STUDIES ON MEASURING SOFTWARE," Report No. IEEE Log Number 8932097. pp.234-246. Order No. 0733-8716/90/0200-0234$01.00.

Dominick, Wayne D.; Moreau, Dennis R.; "OBJECT-ORIENTED GRAPHICAL INFORMATION SYSTEMS : RESEARCH PLAN AND EVALUATION METRICS," In Journal of Systems and Software. 10(1): Jan 1989. pp. 23-28. .

Stockman, Sinclair Guillaume; Todd, Alan Richard Todd; Robinson, Anthony; "A FRAMEWORK FOR SOFTWARE QUALITY MEASUREMENT," Report No. IEEE Log Number 8932093. pp. 224-233. Order No. 0733- 8716/90/0200-0224$01.00.

Leach, Ronald L.; SOFTWARE METRICS ANALYSIS OF THE ADA REPOSITORY. In Proceedings of the 7th Annual National Conference on Ada Technology, March 13-16, 1989. pp. 270-277.

Henry, Sallie; Selig, Calvin; "PREDICTING SOURCE-CODE COMPLEXITY AT THE DESIGN STAGE," In IEEE Software. 7(2): Mar 1990. pp 36 - 43. Order No. 0740-7459/90/0300/0036/S01.00.

Fenick, Stewart; "IMPLEMENTING MANAGEMENT METRICS: AN ARMY PROGRAM," In IEEE Software. 7(2): Mar 1990. pp 65 - 72. Order No. 0740-7459/90/0300/0065/$1.00.

Brandt, Dennis L.; "QUALITY MEASURES IN DESIGN, FINDING PROBLEMS BEFORE CODING," In Software Engineering Notes (ACM SIGSOFT). 15(1): Jan 1990. pp. 68-72.

Munson, John C.; Khoshgoftaar, Taghi M.; "APPLICATIONS OF A RELATIVE COMPLEXITY METRIC FOR SOFTWARE PROJECT MANAGEMENT," In Journal of Systems and Software. 12(3): Jul 1990. pp.283-291. Order No. 0164 1212/90/$350.

Baker, Albert L.; Bieman, James M.; Fenton, Norman; Gustafson, David A.; Melton, Austin; Whitty, Robin; "A PHILOSOPHY FOR SOFTWARE MEASUREMENT," In Journal of Systems and Software. 12(3): Jul 1990. pp. 277-281.

Myrvold, Alan; "DATA ANALYSIS FOR SOFTWARE METRICS," In Journal of Systems and Software. 12(3): Jul 1990. pp. 271-275.

Pfleeger, Shari Lawrence; McGowan, Clement; "SOFTWARE METRICS IN THE PROCESS MATURITY FRAMEWORK," In Journal of Systems and Software. 12(3): Jul 1990. pp. 255-261.

Bhide, Sandhiparakash; "GENERALIZED SOFTWARE PROCESS-INTEGRATED METRICS FRAMEWORK," In Journal of Systems and Software. 12(3): Jul 1990. pp. 249-254.

Lanphar, Robert; "QUANTITATIVE PROCESS MANAGEMENT IN SOFTWARE ENGINEERING, A RECONCILIATION BETWEEN PROCESS AND PRODUCT VIEWS," In Journal of Systems and Software. 12(3): Jul 1990. pp. 243-248.

This page intentionally left blank.

APPENDIX A

Workshop Agenda

WORKSHOP AGENDA

	Monday	**May 21, 1990**
PM	7.:00 - ??	Registration and Reception
	Tuesday	**May 22, 1990**
AM	7:00 - 9:00	Continental Breakfast
	9:15 - 9:45	"Welcome on Behalf of TTCP" Mr. Joseph Batz United States National Leader and Chairperson, XTP-2
	9:45 - 10:00	Administrative Remarks Mr. Jeffrey Lasky Rochester Institute of Technology
	10:00 - 10:15	Break
	10:15 - 11:00	Keynote Address "A Perspective on Software Metrics" Dr. Stephen Yau University of Florida
	11:00 - 11:30	Workshop Charge Mr. Samuel DiNitto Rome Laboratory
	11:30 - 1:30	Lunch
PM	1:30 - 5:00	Panel Sessions
	Wednesday	**May 23, 1990**
AM	8:30 - 10:00	Panel Sessions Continue
	10:00 - 10:15	Break
	10:15 - 12:00	Panel Progress Presentations and Discussions
PM	12:00 - 1:30	Lunch
	1:00 - 4:45	Panel Sessions Continue
	6:00	Group Dinner Michael Cusumano, Guest Speaker

	Thursday	**May 24, 1990**
AM	8:30 - 10:00	Panel Sessions Continue
	10:00 - 10:15	Break
	10:15 - 12:00	Panel Progress Presentations and Discussions
PM	12:00 - 1:30	Lunch
	1:30 -	Final Panel Sessions

This page intentionally left blank.

APPENDIX B

Attendee Directory

ATTENDEES

William Agresti
MITRE Corp
Information Systems &
Technology Division
7325 Colshire Drive
McLean VA 22101-7577
(703) 883-7577

Lou Ayers
WRCA/AAAF
WPAFB OH 45433-6543

Joseph Batz
OUSD(A)/DDDR&E(R&AT)
Pentagon, Rm 3E114
Washington DC 20301-3080
(703) 614-0213

Walter Beam
3824 Forth Worth Ave
Alexandria VA 22304
(703) 370-3431

David Budgen
Dept of Computer Science
University of Stirling
Stirling, Scotland FK94LA
(0786) 73171

Joseph Cavano
Rome Laboratory
RL/C3CB
Griffiss AFB NY 13441-5700
(315) 330-4063

C K S Chong Hok Yuen
College Militaire Royal de Saint-Jean
Quebec, Quebec
Canada JOJ1RO

Andrew Chruscicki
Rome Laboratory
RL/C3CB
Griffiss AFB NY 13441-5700
(315) 330-4476

Judy Clapp
MITRE Corp
A345
Burlington Rd.
Bedford MA 01730
(617) 271-3452

Bob Converse
Computer Sciences Corp
4061 Powdermill Rd.
Calverton MD 20705
(703) 876-1210

Robert Cruickshank
Software Productivity Consortium
2214 Rockhill Rd.
Herndon VA 22070
(703) 742-7116

Samuel A DiNitto
Rome Laboratory
RL/C3C
Griffiss AFB NY 13441-5700
(315) 330-3011

Roger Drabick
Eastman Kodak Company
Hawkeye Plant
20 Avenue E
Rochester NY 14653-7006
(716) 253-5368

Janet Dunham
Herbert Building
PO Box 12194
Research Triangle Part
NC 27709
(919) 541-6562

Roger Dziegiel
Rome Laboratory
RL/C3CB
Griffiss AFB NY 13441-5700
(315) 330-4476

Tom Eller
Kaman Sciences Corp
1500 Garden of the Gods Rd.
Colorado Springs CO 80907
(719) 591-3624

Walter Ellis
IBM
9331 Corporate Blvd
Rockwille MD 20850
(301) 640-2889

Ray Engleman
TRW (FPZ/3224)
1 Federal Systems Park Dr.
Fairfax VA 22033
(703) 968-1249

Stewart Fenick
US Army/CECOM
Center for Software Engineering
AMSEL-RD-SE-AST-SE (Fenick)
Fort Monmouth NJ 07703

Eugene Fiorentino
Rome Laboratory
RL/ERSR
Griffiss AFB NY 13441-5700
(315) 330-3476

Kaye Grau
Software Productivity Solutions
122 North 4th Avenue
Indialantic FL 32903
(407) 984-3370

Jean-Claude Labbe
Defense Research Est., Valcartier
PO Box 8800
Courcelette, Quebec GOA 1RO
Canada
(418)844-4346

Steffan Landherr
IT Division, ERL
PO Box 1600
Salisbury, South Australia 5108
61-8-259-555

Jeff Lasky
Rochester Institute of Technology
One Lomb Memorial Drive
Rochester NY 14623
(716) 475-2284

Kenneth Littlejohn
Wright Laboratory
WL/AAAF-3
WPAFB OH 45433-6543
(513) 255-6548

Michael Looney
Admiralty Research Establishment
Procurement Executive (MOD)
Portsdown
Portsmouth Hants UKPO6 4AA

John McDermott
Naval Research Lab
Code 5542
Washington DC 20375-3770
(202) 767-3770

Rhoda Novak
Aerospace Corp
PO Box 92957
M8-113
Los Angeles CA 90009-2957

John Palaimo
Rome Laboratory
RL/C3CB
Griffiss AFB NY 13441-5700
(315) 330-4477

Robert Park
Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213-3890
(412) 268-5785

Glen Racine
US Army/AIRMICS
115 O'Keefe Bldg.
GIT
Atlanta GA 30332-0800
(404) 894-3110

Carl Seddio
Eastman Kodak Company
Elmgrove Plant
901 Elmgrove Rd.
Rochester NY 14653-6026
(716) 726-9192

Richard Selby
University of California at Irvine
Dept. of Computer Science
Irvine CA 92717

Christopher Smith
Software Engineering Research Center
Georgia Institute of Technology
Atlanta GA 30332-0280
(404) 894-3154

V. C. Sobolewski
Embassy of Australia
1601 Massachusetts Ave., NW
Washington, DC 20036
(202) 797-3378

Paul Szulewski
Draper Laboratory
555 Technology Square, MS 64
Cambridge MA 02139

Thomas Triscari
Rome Laboratory
RL/C3C
Griffiss AFB NY 13441-5700

Ray Walsh
Dynamics Research Corp
Systems Division
60 Frontage Rd.
Andover MA 08110
(508) 475-9090 Ext 1204

Steve Yau
Computer & Information Sciences Dept
301 CSE Bldg.
University of Florida
Gainesville FL 32611

APPENDIX C

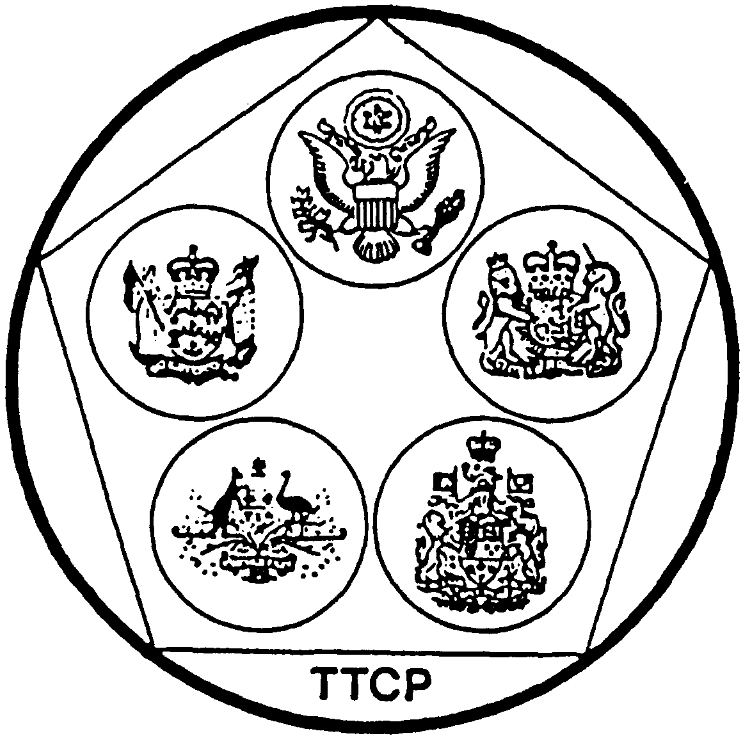
Presentation Slides

Welcoming Remarks

Mr. Joseph Batz

United States National Leader and Chairperson, XTP-2

THE TECHNICAL COOPERATION PROGRAM



MEMBER NATIONS

- AUSTRALIA
- CANADA
- NEW ZEALAND
- UNITED KINGDOM
- UNITED STATES

THE
TECHNICAL
COOPERATION
PROGRAM

FUNCTION

PROVIDE MECHANISMS FOR:

- Science & Technology Information Exchange
- Collaborative Research & Development
- Scientific Personnel Exchange
- Science & Technology Materiel Exchange

QUID PRO QUO

GOVERNMENT TO GOVERNMENT

DEFENSE LIMITED

THE TECHNICAL COOPERATION PROGRAM

JOINT DECLARATION
U.S. President & British Prime Minister

Oct. 25, 1957

"The arrangement which the nations of the free world have made for collective defense and mutual help are based on the recognition that the concept of national self-sufficiency is now out of date. The countries of the free world are interdependent and only in genuine partnership, by combining their resources and sharing tasks in many fields, can progress and safety be found. For our part we have agreed that our two countries will henceforth act in accordance with this principle."

- TRIPARTITE TECHNICAL COOPERATION PROGRAM
  Canada joined U.S. & U.K. immediately
- THE TECHNICAL COOPERATION PROGRAM
  Australia - July 1965
  New Zealand - October 1969

THE
TECHNICAL
COOPERATION
PROGRAM

TTCP AIMS

- PROVIDE KNOWLEDGE & INFORMATION ON EACH OTHERS PROGRAMS
- AVOID UNNECESSARY DUPLICATION AMONG PARTICIPANTS
- PROMOTE CONCERTED JOINT EFFORTS TO CLOSE GAPS

ENCOMPASSING

- BASIC RESEARCH
- EXPLORATORY DEVELOPMENT
- DEMOS OF ADVANCED TECH DEVELOPMENT

THE
TECHNICAL
COOPERATION
PROGRAM

TECHNOLOGY AREAS

SUBGROUPS

- Chemical Defense
- Aeronautics Technology
- Radar Technology
- Electronic Warfare
- Behavioral Sciences

- Undersea Warfare
- Infrared & ElectroOptical Technology
- Materials
- Communications Technology & Information Systems
- Conventional Weapons Technology

- COMPUTING TECHNOLOGY

PANELS; ACTION GROUPS; TECH LIAISON GROUPS

THE
TECHNICAL
COOPERATION
PROGRAM

COMPUTING TECHNOLOGIES SUBGROUP X (SGX)

TECHNICAL PANELS

XTP1	-	Trustworthy Computing
XTP2	-	Software Engineering
XTP3	-	Architectures
XTP4	-	Machine Intelligence

ACTION GROUPS

XAG2	-	Digital Design
XAG3	-	Image Information

THE
TECHNICAL
COOPERATION
PROGRAM

XTP2 - SOFTWARE ENGINEERING

PURPOSE: To improve the utilization of the collective resources and capacities of the member countries in the areas of software engineering and software technology.

SCOPE: The creation and life-cycle support of software for military applications.

Includes: PROCESSES, METHODS, TOOLS for

DEFINITION	SPECIFICATION	PROTOTYPING
DESIGN	INTEGRATION	TEST
EVALUATION	PORTING	REUSE

DATABASE TECHNOLOGY

THE
TECHNICAL
COOPERATION
PROGRAM

XTP2 - WORKSHOPS

- REAL TIME SYSTEMS AND ADA
  Conducted June 1988, at IDA, Washington DC.
  Approx. 40 participants.
- REQUIREMENTS ENGINEERING/RAPID PROTOTYPING
  Planned for November 1989, at Fort Monmouth
  (Eatontown), NJ.
- SOFTWARE METRICS
  Planned in 1990.

Keynote Address

Dr. Stephen Yau

University of Florida

A Perspective on Software Metrics

Stephen S. Yau

Software Engineering Research Center
Computer and Information Sciences Department
University of Florida
Gainesville, Florida

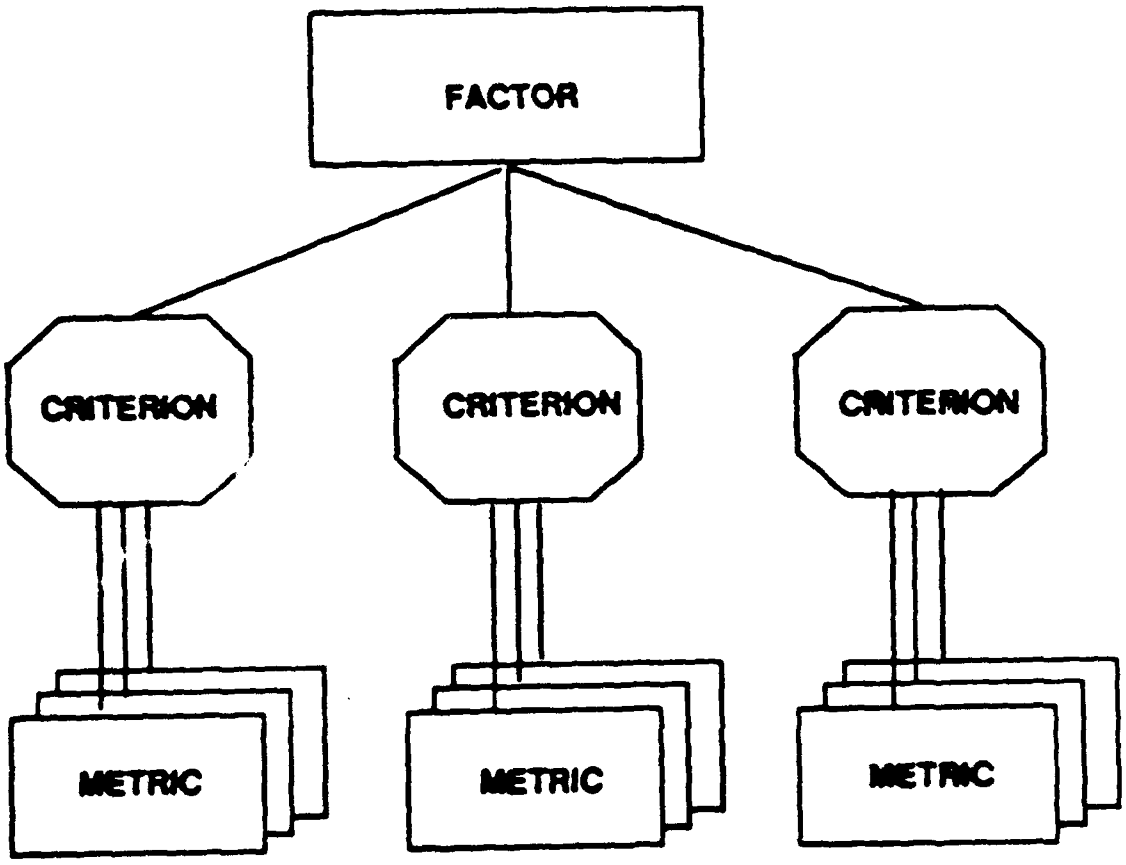
May 22, 1990

Examples for Using Software Metrics

- Rombach's Design Measurement Framework for Software Design.
- Grady's Work-Product Analysis for Software Development.
- Henry and Selig's Predicting Source-Code Complexity for Software Design.
- Porter and Şelby's Metric-Based Classification Trees for Software Development.
- Enrich's Reliability-Measurement Approach for Software Test.
- Fenick's Management Metrics for the Software Development Life Cycle.
- Ramamoorthy's Metrics Guided Methodology for Software Requirement.
- Miyoshi's Formal Approach to Software Environment Technology for Software Development Environment Evaluation.

- Goals for using software metrics:
  - Achieving quality assurance.
  - Improving software productivity.
- Development of effective metrics
- Validation:
  - Techniques
  - Data collection
  - Analysis
- Applications

- Define the framework of metrics.
- Develop the necessary metrics and collect the data.
- Modify the development activities to improve software quality and productivity.



Software Quality Hierarchy

Software Quality Hierarchy

- Factors: User/management-oriented terms. They represent the characteristics which comprise the software quality.
- Criteria: Software-related terms. They are the attributes that provide the characteristics represented by the quality factors.
- Metrics: Quantitative measures of the software attributes defined by the criteria.

The weight of a metric or a criterion can be increased or decreased according to the user's preference.

Software Quality Factors

- Reliability - Extent to which the software will perform without any failures within a specified time period.
- Usability - Relative effort training on software operation.
- Security - Extent to which the software will perform without failures due to unauthorized access to the code or data within a specified time.
- Survivability - Extent to which the software will perform and support critical functions without failures within a specified time period when a portion of the system is inoperable.
- Efficiency - Relative extent to which a resource is utilized.
- Maintainability - Ease of effort for performing any maintenance activities.
- Flexibility - Ease of effort for changing the software missions, functions, or data to satisfy other requirements.
- Interoperability - Relative effort to couple the software of one system to the software of another system.
- Portability - Relative effort to transport the software for use in another environment.
- Reusability - Relative effort to convert a software component for use in another application.

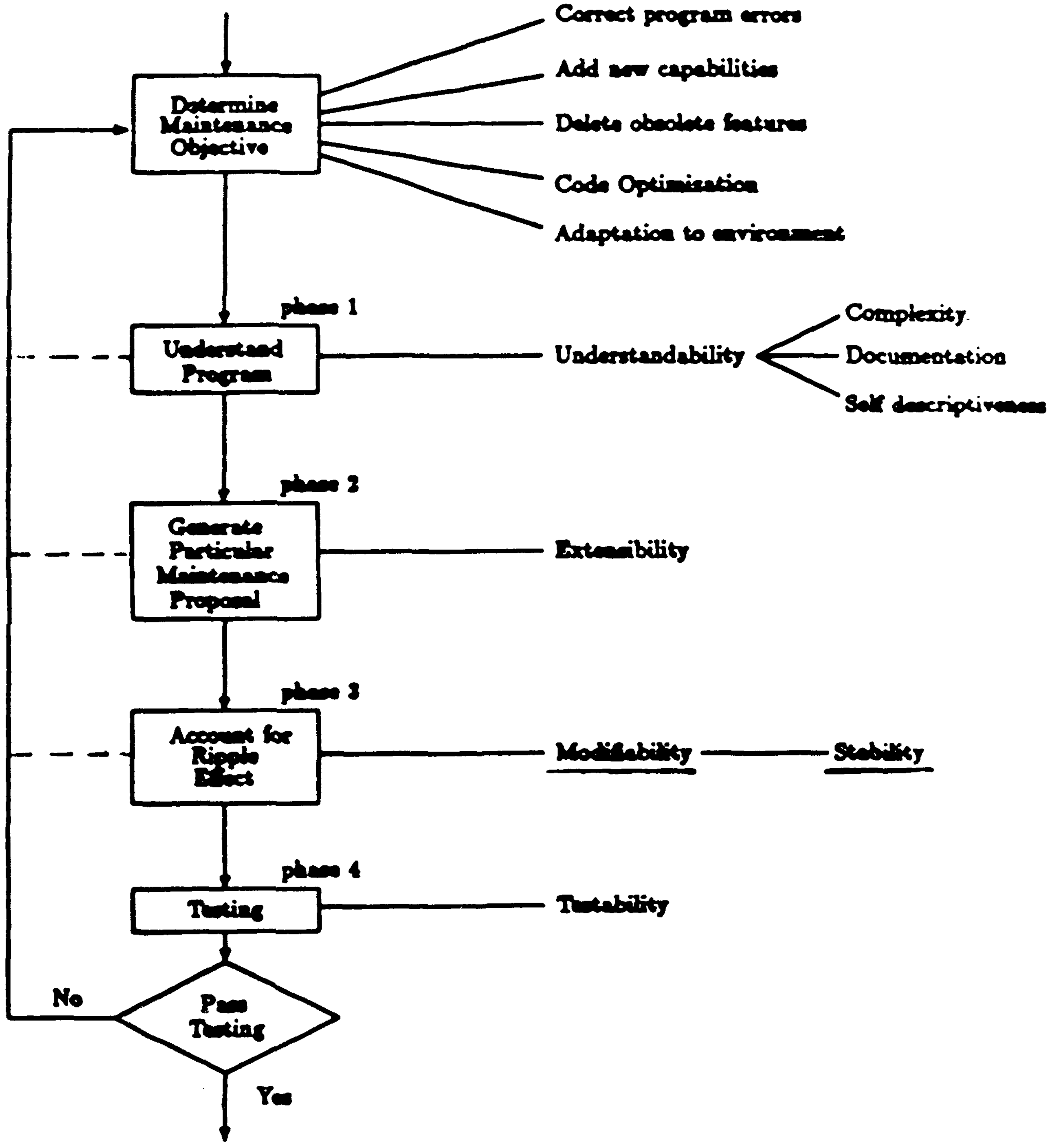


Fig. 1 The software maintenance process and associated program quality attributes in each phase.

STABILITY MEASURES

Stability of the software should be achieved by minimizing potential ripple effect caused by interaction between modules (i.e., a change to a module causes undesirable changes to other modules).

- Program Stability – Quality attribute indicating the resistance to the potential ripple effect which a program would have when it is modified.
- Module Stability – A measure of the resistance to the potential ripple effect of a modification of the module on other modules in the program.
- Logical Stability – A measure of resistance to impact of modification on other modules in terms of logical considerations.
- Performance Stability – A measure of resistance to impact of modification on other modules in terms of performance considerations.

LOGICAL STABILITY METRIC

First, we define the logical stability of a module k, denoted by LS_k, as follows:

$$LS_k = 1/LRE_k \tag{1}$$

where LRE_k is the logical ripple effect measure of a primitive type of modification to a module k, where a primitive type of modification is considered as a modification of a variable definition in module k.

$$LRE_k = \sum_{i \in V_k} [P(k_i) * LCM_{k_i}] \tag{2}$$

$V_k =$ the set of all variable definitions in module k,

$P(k_i) =$ the probability that a particular variable definition of mudule k will be selected for modification,

$LCM_{k_i} =$ the logical complexity of each modification to variable definition i in module k

$$LCM_{K_i} = \sum_{t \in W_{k_i}} C_t \qquad (3)$$

C_t = the complexity measure of module t

W_{k_i} = the modules involved in the intermodule change propagation as a consequence of modifying variable definition i of module k

$$W_{k_i} = \bigcup_{j \in Z_{k_i}} X_{k_j} \qquad (4)$$

Z_{k_i} = the set of interface variables which are affected by logical ripple effect as a consequence of modification to variable i in module k,

X_{k_j} = the set of modules involved in intermodule change propagation as a consequence of affecting interface variable j of module k.

Normalized Stablility Measure

The logical stability measure may be normalized to have a value ranging from 0 to 1, with 1 being the optimal logical stability. This normalized logical stability can be utilized qualitatively or it can be correlated with collected data to provide a quantitative measure of stability. The normalized logical stability measure for module k, denoted by LS_k^*, is defined as follows:

$$LS_k^* = 1 - LRE_k \tag{7}$$

where LRE_k is the normalized logical ripple effect measure for module k

$$LRE_k = LRE_k^+ / C_p \tag{8}$$

C_p = the total complexity of the program which is the sum of all the module complexities in the program.

LRE_k^+ = the modified logical ripple effect measure for module k

$$LRE_k^+ = C_k + \sum_{i \in V_k} [P(k_i) * LCM_{k_i}] \tag{9}$$

Finally, we define the logical stability of a program, denoted by LSP, as follows:

$$LSP = 1/LREP \qquad (10)$$

where $LREP$ is the measure of the potential logical ripple effect of a primitive modification to a program:

$$LREP = \sum_{k=1}^{n} [P(k) * LRE_k] \qquad (11)$$

$P(k)$ = the probability that a modification to module k may occur

LRE_k = the logical ripple effect measure of a primitive type of modification to a module k

n = the number of modules in the program.

Normalized Stability Measure :

The normalized logical stability measure of a program, denoted by LSP^*, is defined as follows:

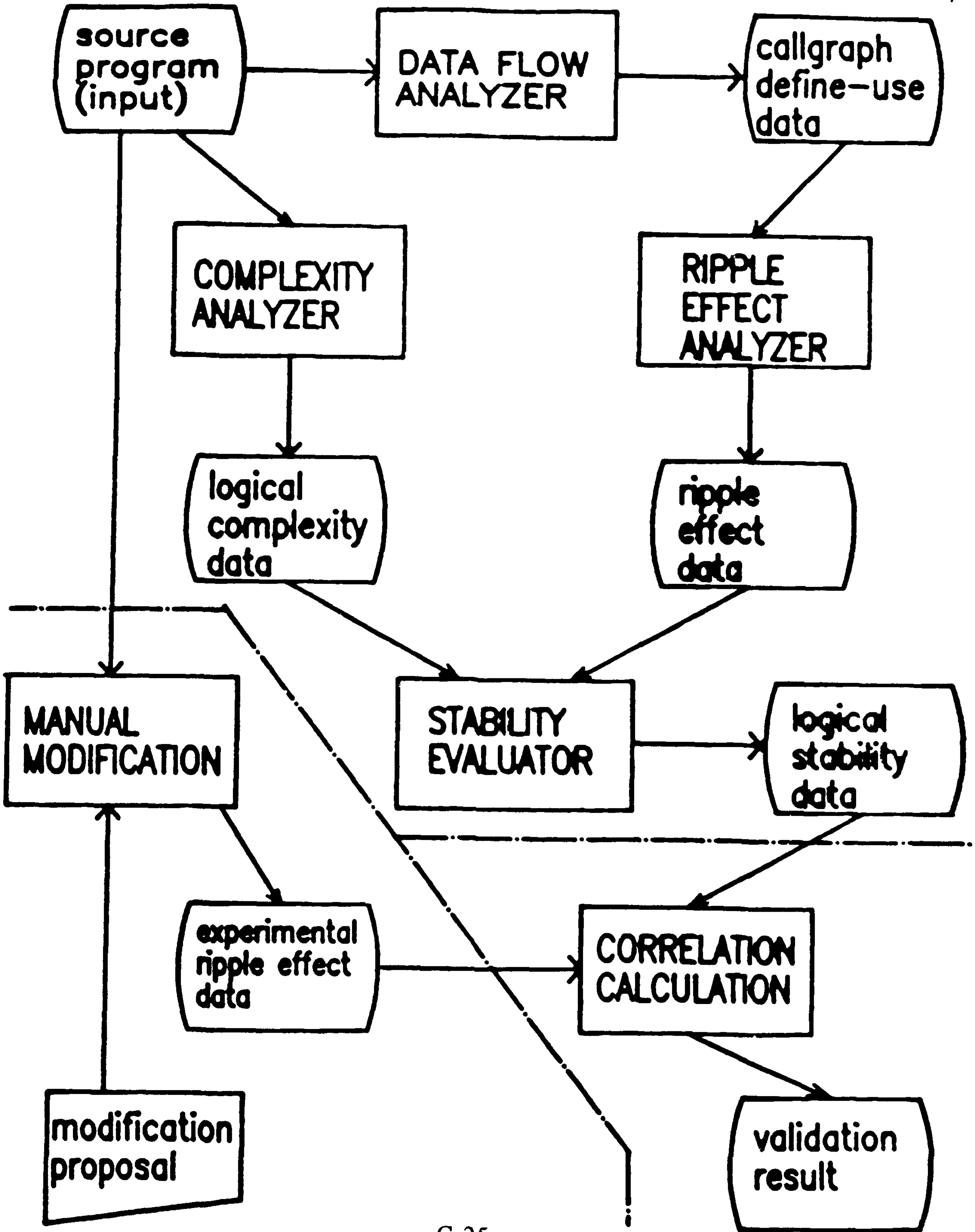
$$LSP^* = 1 - LREP^* \qquad (12)$$

where $LREP^*$ = the normalized logical ripple effect measure of the program

$$= \sum_{k=1}^{n} [P(k) * LRE_k^*] \qquad (13)$$

$P(k)$ = the probability that a modification to module k may occur

LRE_k^* = the normalized logical ripple effect measure for module k.



SYSTEM BLOCK DIAGRAM

VALIDATION SOFTWARE STABILITY METRICS

- Methods to Conduct Validation Experiments

- Token Count Method

  Measure the token* changes resulting from the logical Ripple Effect of a primitive modification to a token in the initially modified procedure(s).

- Time Measurement Method

  Measure the time required to maintain the semantic consistency of the whole program after a primitive modification** is performed to a particular module in the program.

- Statement Count Method

  Measure the statement*** changes resulting from the logical ripple effect of a primitive modification to a simple statement in the initially modified procedure(s).

* Token - A basic recognizable unit in the lexical analysis phase of the program compiling process (i.e., keyword, operator, etc.).

** Primitive Modification - The modification to a definition of a variable.

*** Statement - The smallest program segment which is separately executable

DATA ANALYSIS

The experiment result

program name		GSDSI	GSDSII	LIBI	LIBII	ACCTI
no. of lines		14,000	7,000	11,000	7,500	10,000
no. of modules in the program		281	118	282	145	167
no. of modification proposals		40	40	35	35	30
no. of modules modified		103	71	60	54	50
estimated logical stability average		0.5680	0.3837	0.5833	0.8350	0.6147
experiment result average		0.5052	0.7383	0.5088	0.6736	0.6556
correlation between logical stability and experiment result	token count	0.2535	0.2541	0.5038	0.1288	0.2536
	statement count	0.1588	0.2352	0.4697	0.0622	0.2780
	time measure.	0.1711	0.3168	0.2682	0.1088	0.2650
probability that the actual correlation being zero is less than	token count	10%	10%	5%	15%	10%
	statement count	20%	10%	10%	40%	10%
	time measure.	20%	5%	10%	30%	10%

Effect of Various Maintenance Type on the Validation Experiments

* Purpose:

Different maintenance types usually lead to different program modification style, which in turn, may cause different degrees of logical ripple effect. The purpose of the analyses is to explore the impact of different maintenance types on the cause of logical ripple effect. The results may give us some insight of whether it is suitable to use our logical stability metrics to predict the actual effort required to modify the program in order to carry out certain types of maintenance specifications.

* Classification of Maintenance Specifications

Summary of the Classification of Maintenance Specifications

program name	Number of Speci.	Require. Change	Function Enhance.	Error Correc.	Code Optimiz.	Deletion of Feature
GSDS I	40	7	26	8	0	0
GSDS II	40	1	26	6	7	0
LIB I	35	9	26	0	0	1
LIB II	35	15	20	0	0	0
ACCT	30	3	11	14	2	1

Software Development Process

- The *process* of software creation and evolution is as important as the *product* of that process.
- Processes are software too - They are created in order to describe the way in which specific information products are to be systematically developed or modified.
- The process should be evolutionary.
- The process should be formalized and automated.
- **The software quality depends on the process.**

Software Process Factors

- Software processes are software, too. There is a life cycle of software processes corresponding to the life cycle of the development of software products. Therefore, software process quality shares similar quality factors as in the software product quality. We also include the followings:
- Structurability - Extent to which the software development process can be break down into manageable sub-processes.
- Instantiablity - Extent to which the software process can be instantiated with the current development environment.

11

IEEE Computer Society Standards Activities

- Software Quality Standards.
- Software Productivity Standards.

$$\text{Productivity} = \frac{\text{Output product}}{\text{Input effort}}$$

* Output primitives are source statements, function points and documents

* Input primitives are the effort in terms of staff-hours applied to develop software products

$$\text{Productivity}_a = \frac{O_a}{E_a}$$

$$\text{Productivity}_{final} = \frac{O_{final}}{E_{final}}$$

$$\begin{array}{c}\text{Incremental}\\ \text{Productivity}\end{array} = \frac{O_{t_n} - O_{t_{n-1}}}{E_{t_n} - E_{t_{n-1}}}$$

<u>Output primitives:</u>

* <u>Source statements</u>
  Counting
    Logical source statements
        instructions

    Physical source statements
        lines of code
  Types attribute
    Executable
    Data declaration
    Compiler directive
    Comments
  Origin attribute
    Developed (new and modified)
    Non-developed (deleted and reused)
  Usage attribute
    Delivered
    Non-delivered (support development)

Output primitives (cont.)

* Function points

Counting

computed from an algorithm using various characteristics of the application's functionality expressed as a relationship of function type.

Type attribute

* Document

Counting

"page" or "screen"

- Page count
- Page size
- Token count
  - . words
  - . ideograms
  - . graphics, graphs, tables, figures, charts, pictures

Attributes

Type (name, purpose)

Origin (reused / total (hi-level))

Usage (delivered & non-delivered)

<u>Input primitives</u>

* Staff Traits
* Staff effort attributes
  - Direct
  - Support

Characteristics

* Project
  - Personnel (education, experience, expert assistance, training, size, turnover)
  - Software development environment (tools, techniques, management e.g. PERT chart, critical path analysis)
* Management
  - User participation
  - Stability of product requirements
  - Constraining management factors
* Product
  - Criticality (timing, memory, quality/reliability)
  - Degree of innovation
  - Complexity (programming, data, organizational)
* Concurrency (concurrent development of SW, HW, FW)
* Product Description

Useful Metrics

- Significant to specified goals. (overall framework)
- Consistent with intuition. (common sense)
- Easy to measure and compute.
- Easy to validate. (updated as more data is collected)

Easy to use!

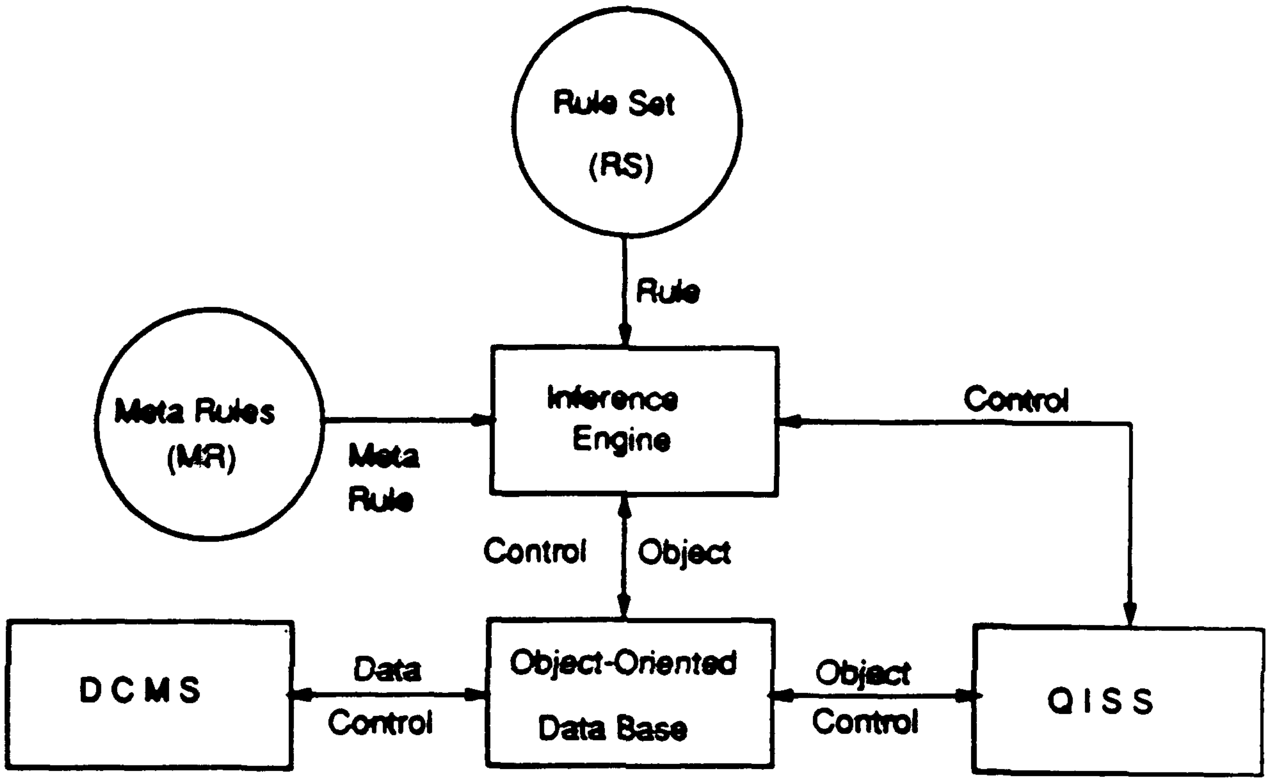
Software Quality Assurance Expert System Framework

Expert System:

- Knowledge base.
  - Inferential knowledge : Rule Set, Meta Rules, and QISS.
  - Facts: Object-Oriented Data Base.
- Inference engine.

Components of the Expert System Framework:

- Data Collection and Measurement System (DCMS).
- Object-Oriented Data Base.
- Rule Set (RS) and Meta Rules (MR).
- Quality Improvement Suggestion System (QISS).



DCMS: Data Collection and Measurement System

QISS: Quality Improvement Suggestion System

Software Quality Assurance Expert System Framework (SQAESF)

Processes of SQAESF

Step 1. DCMS acquires the actual quality metric values of the software product and users' expected quality factor values from developers and store the data in the Object-Oriented Data Base.

Step 2. Based on the rules of MR and RS, the Inference Engine provides quality improvement suggestions. The rules were generated based on the relationships of factors, criteria, and metrics.

Step 3. The Inference Engine compares the actual quality factor values with users' expected quality factor values. If the former is smaller than the latter, the Inference Engine invokes QISS and goto **Step 4**, otherwise goto **Step 6**.

Step 4. QISS invokes the dependency-directed backtracking method to find the metrics that cause the low quality.

Step 5. Based on the metric elements found in **Step 4**, the Inference Engine asks developers to modify the design.

Step 6. Stop.